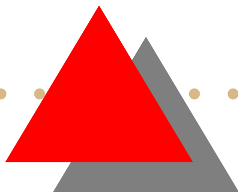
Event-based Coordination of Process-oriented Composite Applications

Marlon Dumas, Tore Fjellheim, Stephen Milliner

Queensland University of Technology, Australia

Julien Vayssiere

SAP Research Centre, Brisbane, Australia



Problem statement

A process-oriented composite application aggregates functionality from other applications following a process model.

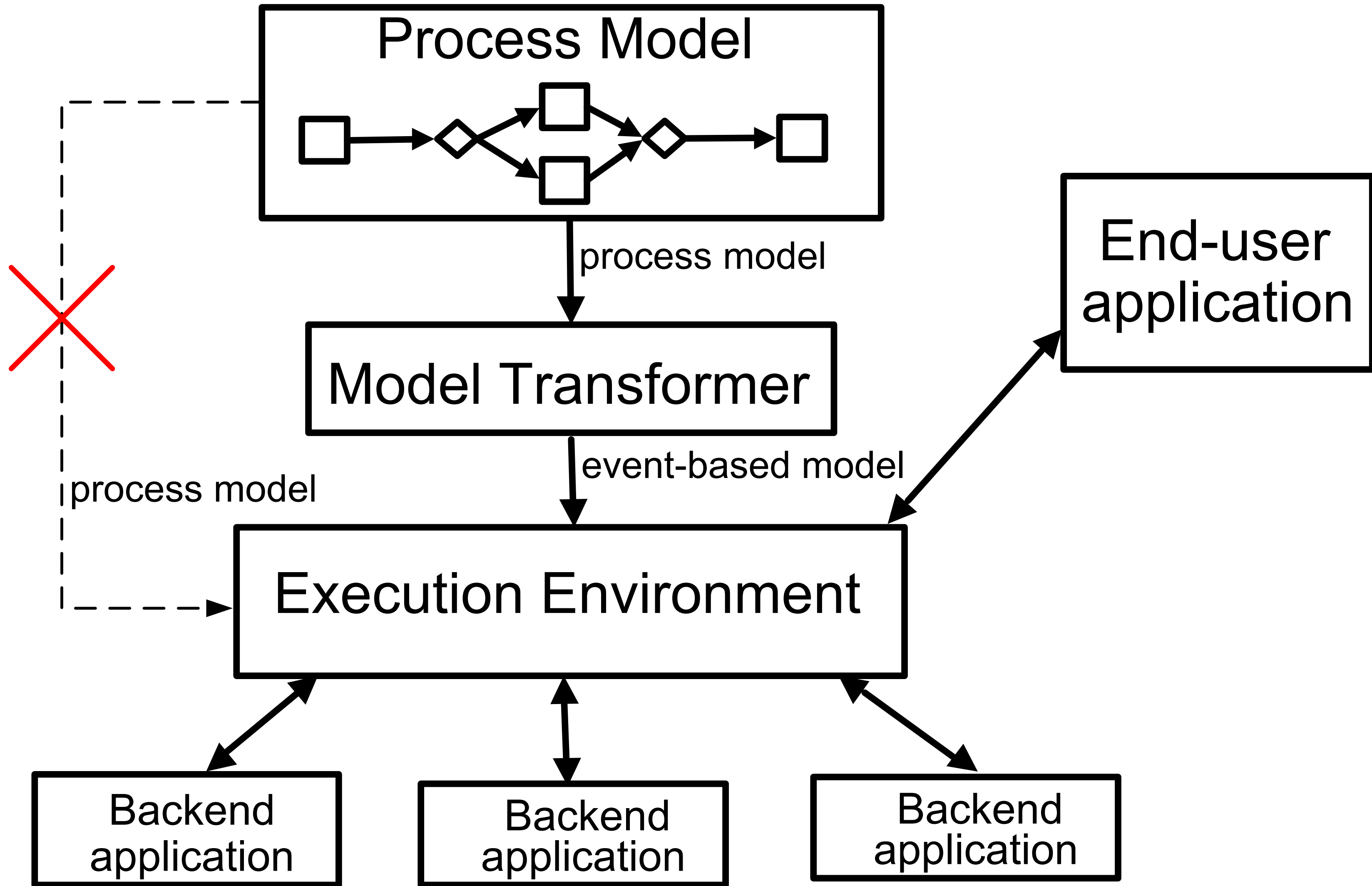
A process-oriented composite application aggregates functionality from other applications following a process model.

Approaches to develop such applications are based on static definition/deployment of models.

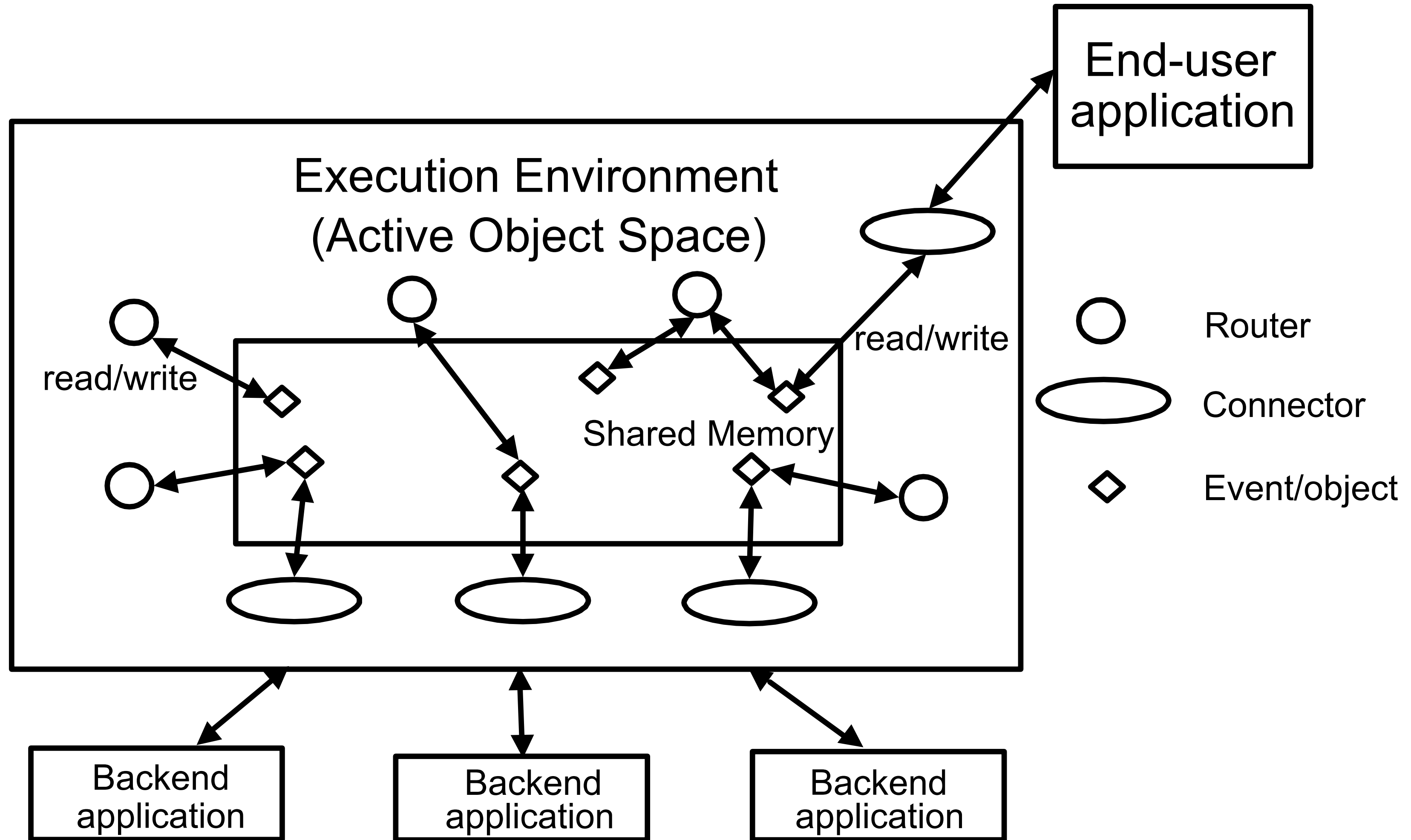
They lack flexibility to support:

- Personalisation
- Context adaptation
- Hot-fixing

Proposed architecture



Execution environment

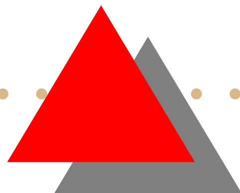


Execution environment (cont.)

Object space. Shared memory space supporting four operations:

- *read* an object matching a template
- *take* an object matching a template
- *write* an object
- subscribe to the arrival of an object matching a template (*notify*)

Object template. Class name + property equality constraints, e.g. *Invoice[clientID=X]*

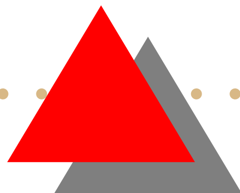


Execution environment (cont.)

Active object. Object with own thread of control

Coordinator. Active object operating in a continuous loop with three steps:

- Wait for arrival of a certain combination of objects to the space
- Perform internal actions and/or interact with external apps
- Write one or multiple objects to the space



Execution environment (cont.)

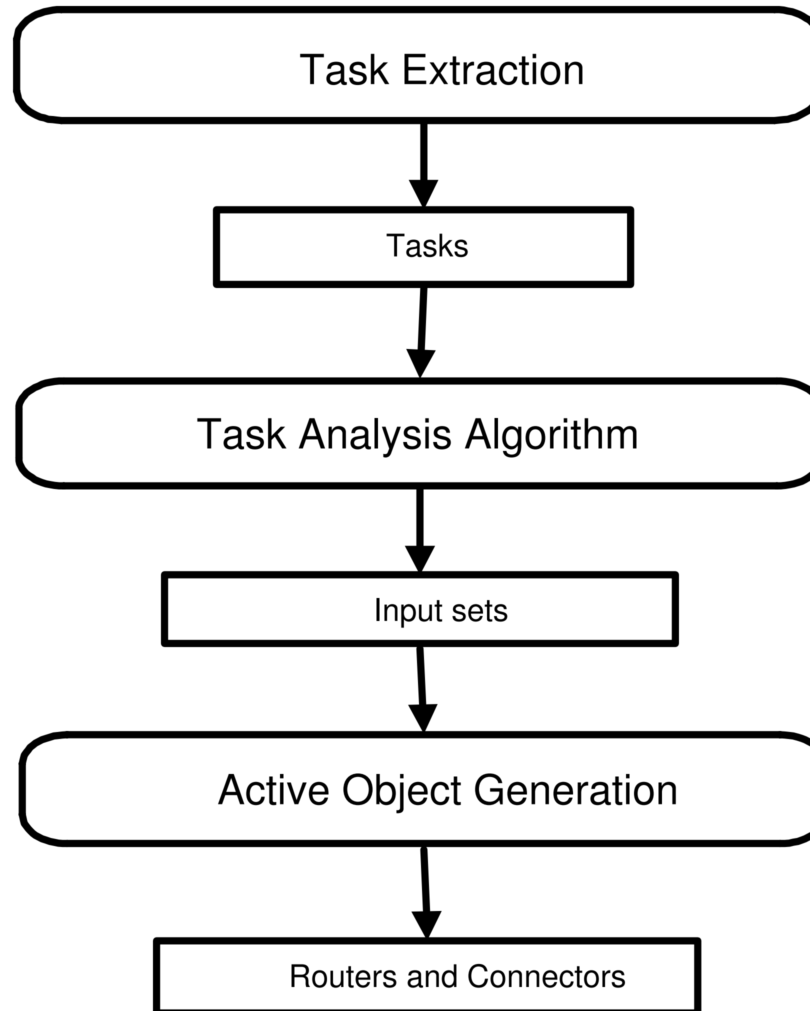
Types of coordinators:

- **Connectors**: Interact with external apps
- **Routers**: Guide the execution of tasks

A router is defined by:

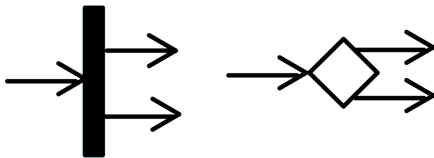
- Input set: object templates and conditions determining when should a task start
- Output: objects produced when task completes
- Stop set: object templates and conditions that desactivate the router

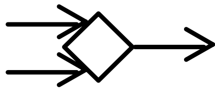
Generating routers

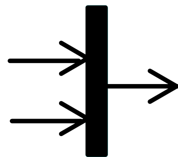


Algorithm for input sets generation

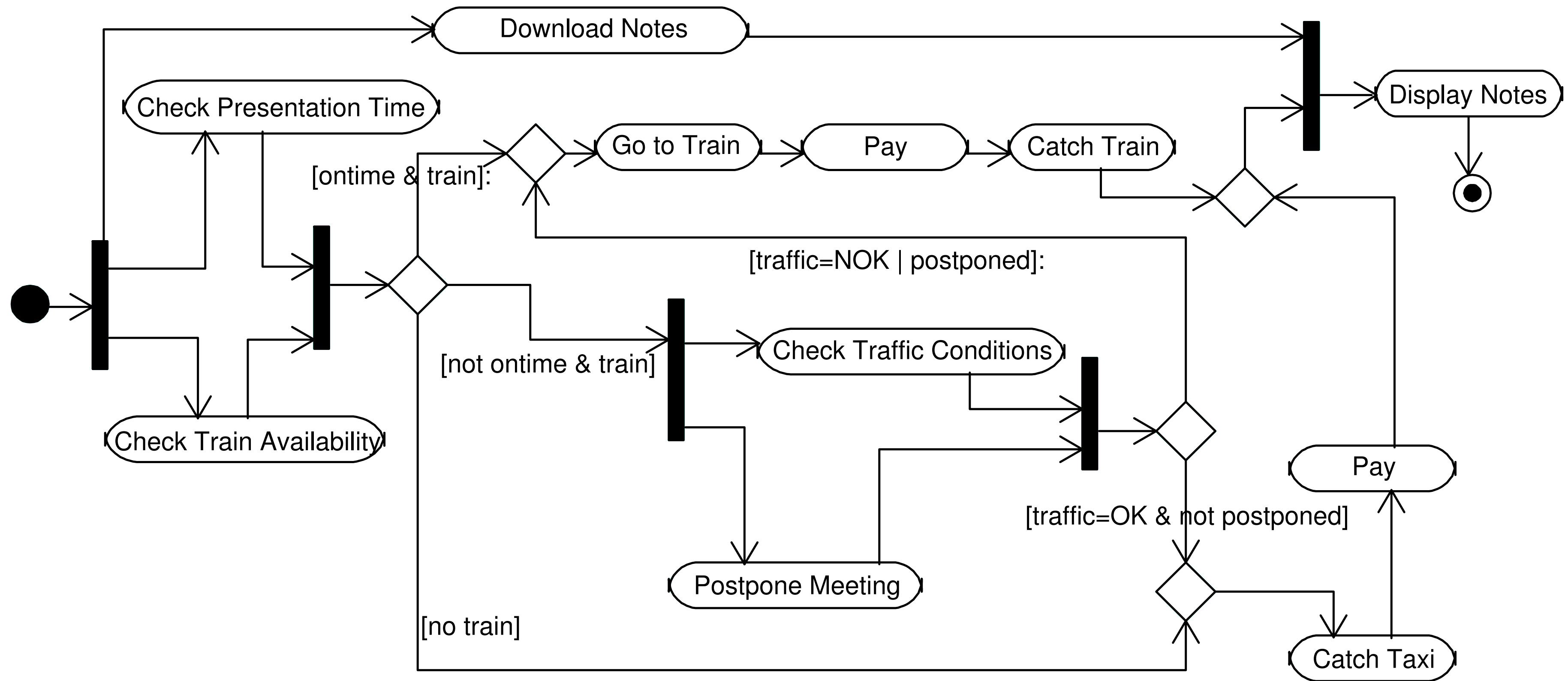
```
InputSetsTrans(t : Transition) :
    let x = Source(t)
        if NodeType(x) = "action"
            return CompletionObject(x)
        else if NodeType(x) = "initial"
            return ProcessInstantiationObject(Process(x))
        else if NodeType(x) in {"decision", "fork"}
            Computer input sets of incoming transition
            Combine input sets with guards (if any)
        else if NodeType(x) = "merge"
            Generate input sets for incoming transitions
            return union of these input sets
        else if NodeType(x) = "join"
            Generate input sets for incoming transitions
            Compute cartesian product of these sets
            Combine resulting vectors in a pointwise manner
```







Process model example



Input set examples

- InputSets(CheckTraffic) =

  { { CompletionObject[action=CheckPresentationTime],
  CompletionObject[action=CheckTrainAvailability],
  not onTime & trainAvailable }

Input set examples

- InputSets(CheckTraffic) =

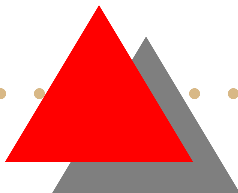
  { { CompletionObject[action=CheckPresentationTime],
  CompletionObject[action=CheckTrainAvailability],
  not onTime & trainAvailable }

- InputSets(GoToTrain) =

  { { CompletionObject[action=CheckPresentationTime],
  CompletionObject[action=CheckTrainAvailability],
  onTime & trainAvailable },
  { CompletionObject[action=CheckTraffic],
  CompletionObject[action=PostponeMeeting],
  traffic = NOK | postponed } }

Execution of a process instance

- End-user application connector writes instantiation object

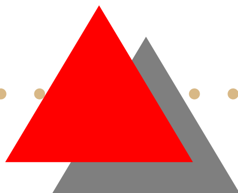


Execution of a process instance

- End-user application connector writes instantiation object
- Routers of CheckPresentationTime, CheckTrainAvailability, DownloadNotes detect object, write task-enabling objects

Execution of a process instance

- End-user application connector writes instantiation object
- Routers of CheckPresentationTime, CheckTrainAvailability, DownloadNotes detect object, write task-enabling objects
- Connectors detect task-enabling objects, perform task, place task completion objects

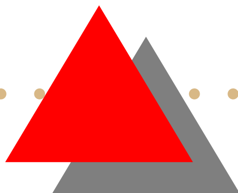


Execution of a process instance

- End-user application connector writes instantiation object
- Routers of CheckPresentationTime, CheckTrainAvailability, DownloadNotes detect object, write task-enabling objects
- Connectors detect task-enabling objects, perform task, place task completion objects
- Task completion objects detected by routers of GoToTrain, CheckTraffic, etc.
- . . .

Consider the following variant of this process:

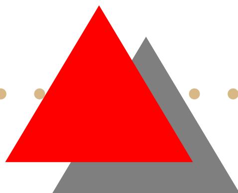
- The execution of task “postpone meeting” takes more time than expected.
- User does not wish to be delayed by this task.
- Instead, if traffic conditions are found to be OK $\longrightarrow$ catch a taxi



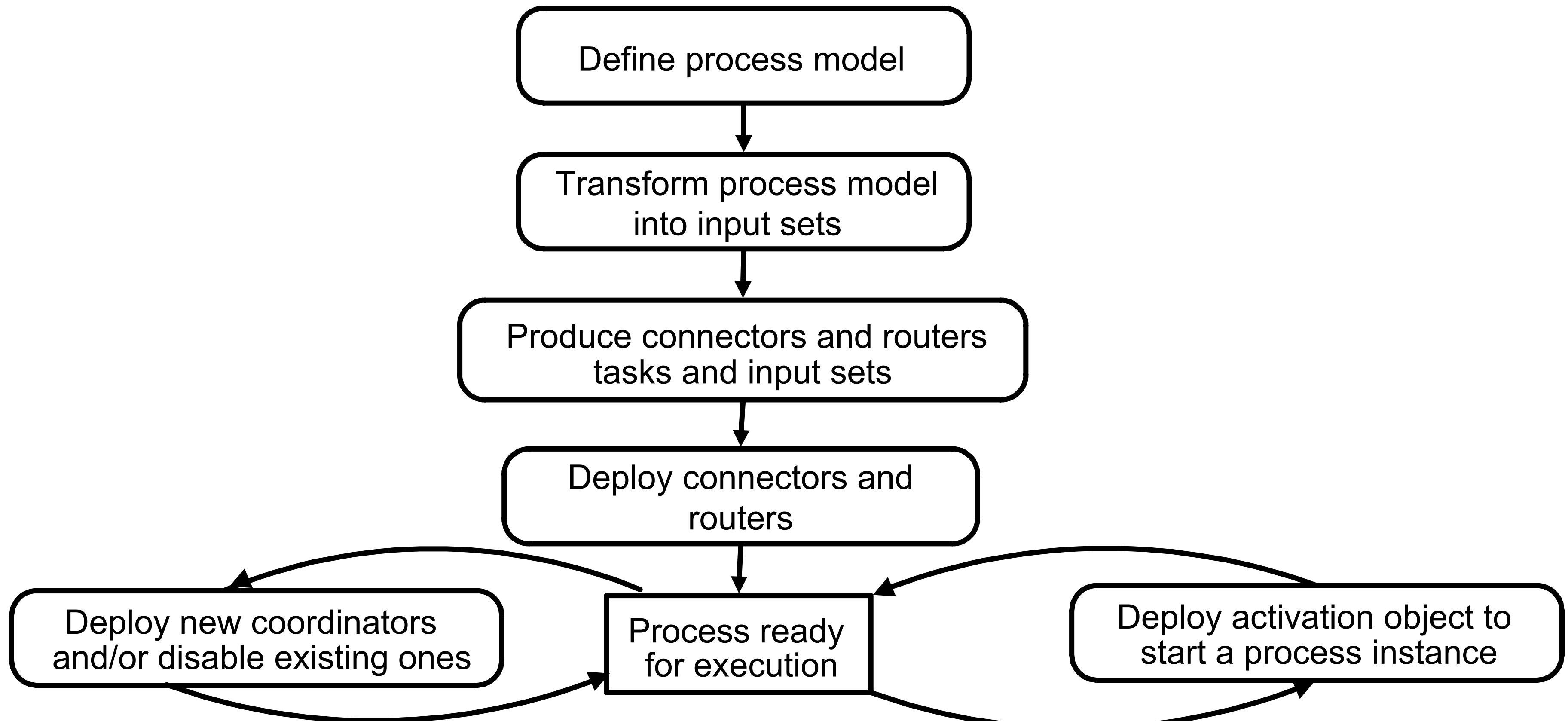
Achieving adaptation (cont.)

User selects an option resulting in the activation of a “short-circuiting” router as follows:

- InputSet =
  { CompletionObject[action=CheckTraffic]
  traffic = OK }
- Output = {EnablingObject[action=CatchTaxi]}
- StopSet =
  { CompletionObject[action=PostponeMeeting] }



Summary of the approach



Future work

- Test proposal against various adaptation scenarios (e.g. context adaptation, hot-fixing)
- Adapt algorithm to translate from global to local process views (e.g. UML AD to WSMO)
- Develop technique for reverse transformation: event-based model $\longrightarrow$ process model