Specifying and Enforcing Intertask Dependencies

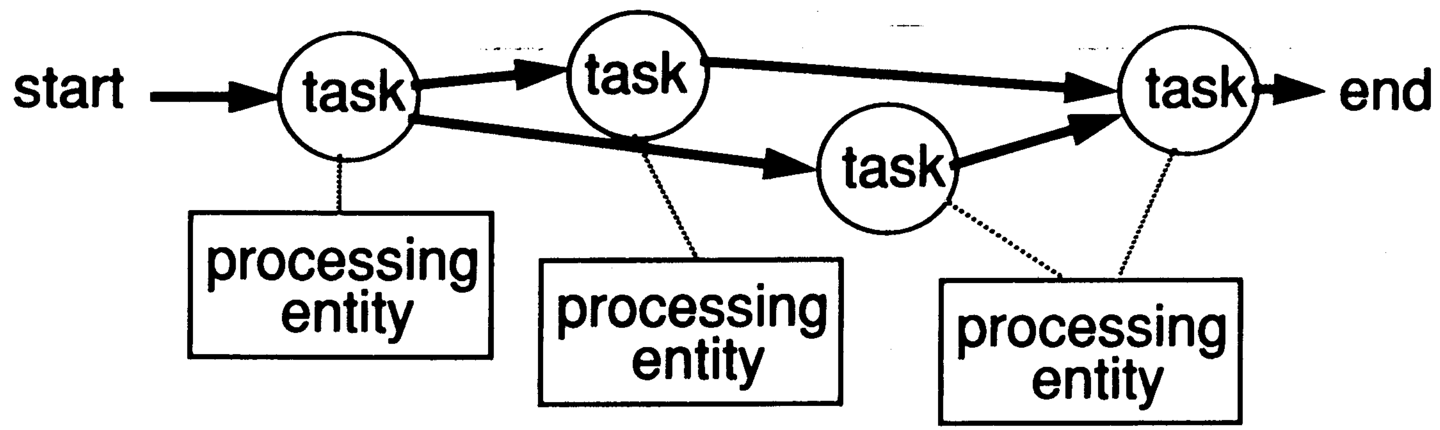
P. Attie, M. Singh
CARNOT, MCC

A. Sheth
Bellcore

M. Rusinkiewicz
U. of Houston

Talk Outline

- Background on workflow management
- Specification of workflow components:
  - tasks
  - formal model for specification of intertask dependencies
- One approach to scheduling
- Work in progress



'hat is a (transactional) workflow?

:hat involves coordinated execution of multiple :s (of different types) by processing entities (of dif- ;).

n issues:

al Tasks:

'ormat: message, contract, form, transaction

structure: externally visible states of the task, l state, termination states, transitions (significant ts) and *their attributes*

operation) semantics, e.g., compatibility, relaxed ion

processing

al Entities:

of entity: human, application system, DBMS

m properties/semantics, e.g., isolation granularity, preservation, idempotency, monotonicity

What is a (transactional) workflow?

- Task Coordination requirements:
  - **Intertask dependencies** and data exchange
- Intra- and inter-workflow Execution requirements:
  - failure atomicity (A)
  - execution atomicity (I)
  - workflow recovery
  - inter-workflow concurrency

Workflow Examples

Environment	Application
office computing	mail routing loan processing meeting scheduling course organizing
data processing	processing a purchase order
manufacturing	product life-cycle
telecommunication	**establishing or changing a service/circuit**

Closely related terms/issues:
Multi-system applications [Bellcore/UofH], task flow [Dayal], long-running activities [DEC], application multi-activities [Kalinechenko], extended transaction models [Elmagarmid book], third generation TP monitor [SIGMOD93]

Related research areas [different types of tasks, different types of entities]: cooperative activity [Bellcore,..], collaborative distributed problem solving [UFL,...], DAI [DAKE, MCC,..], learning, self-adapting software agents [CMU,..]

Transactional Workflow Management

Three Components:

- Specification:

  (a) specification of tasks, (b) dependencies, and
  (c) execution requirements

- Scheduling:

  safe, correct, optimal/efficient, failure handling;
  exploit task and system semantics

- Executing:

  manage execution of tasks/transactions on heterogeneous, autonomous component systems

Related Work

- ACID transactions and their nested derivatives
  Problems: inflexible, difficult to implement in multi-systems.
- Queued message systems and “chaining of transactions”.
  Problems: insufficient control over transaction properties, one type of task, interactions among concurrent activities difficult.

- Extended/Relaxed Transaction Models:
  Sagas and Nested Sagas [Garcia-Molina et al. 88, 90], ConTracts [Reuter 89]. Flexible Transactions [Elmagarmid et al 90, Rusinkiewicz et al 90], Multi-transaction Activities [Garcia-Molina et al. 90], Open Nested Transactions [Weikum & Schek 92] and Others (e.g., in [Elmagarmid 92]), ACTA framework [Chrysanthis & Ramamritham 91/92]

- “Workflow” and hybrid models:
  Long-Running Activities [Dayal et al. 91], DOM model [Buchmann et al 92],
  Georgakopoulos et al
  Third Generation TP Monitors [Dayal et al. 93]

[Dayal 93, Rusinkiewicz & Sheth 93]

Going beyond

Many types of (intertask, multidatabase) dependencies have been defined.

- Lack of formal specification
- Lack of specifications that are executable/postulative
- Correct and safe execution of workflow *wrt* to intertask dependencies.

These issues are addressed in this paper.

Allowable intertask specifications are quite powerful because

- different types of tasks can be modeled and
- intertask dependencies can be associated with transitions (at least one of transitions should be scheduler controllable).

w Specification Model (partial)

uctures

s:

t representation (state transition diagram) of the
: that hides irrelevant details of internal
on of the task

nts [$e, e_i, e_j, .$]:

or) transitions associated with a task agent

ndencies [$d(e_1, \ldots, e_n)$]:

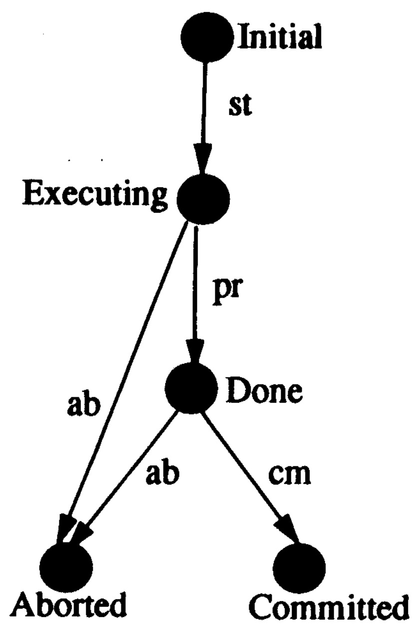
on occurrence and temporal order of task

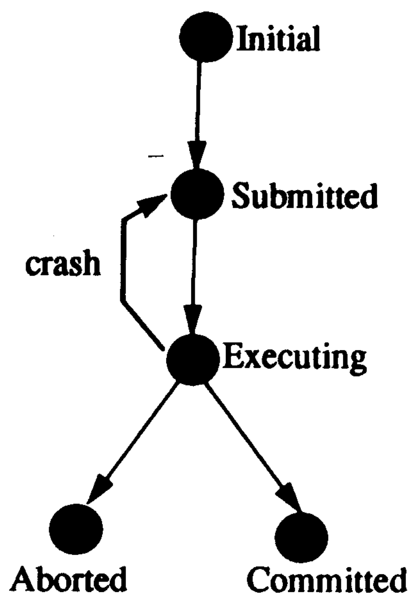
[Attie el al 93]

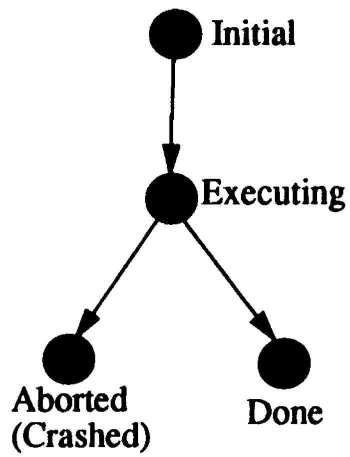
Task Skeleton

- different state transition diagrams for different types of tasks (depending on the application and/or the processing)
  - different states (e.g., no precommit)
  - different significant events (state transition requests) submitted by the task agent

Examples:







[Rusinkiewicz & Sheth 93]

Significant Events

Significant event (task transition request) types for database applications/transactions: *st, ab, pr, cm*

Assume that a "scheduler controls significant event requests" (transition requests).

Possible attribute of a significant event for a "scheduler":

- Forcible: the "scheduler" can always force the event (corresponding execution is guaranteed to occur)
- Rejectable: the "scheduler" can reject the event request and prevent corresponding execution
- Delayable: the "scheduler" can delay the event

Event	Forcible?	Rejectable?	Delayable?
cm	N	Y	Y
ab	Y	N	N
pr	N	N	N
st	Y	Y	Y

Usual attribute assignments for transactions in database applications and DBMSs

* program abort, precommit do not go through scheduler

[Attie et al 93]

Intertask Dependencies

Preconditions for initiating each scheduler-controllable transition in a task.

Klein's primitives [KL91]:

- Order Dependency: $e_1 \overset{t}{<} e_2$.
  If both e_1 and e_2 occur, then e_1 precedes e_2.
  Alternatively, in CTL: if e_2 occurs, e_1 cannot occur subsequently.
  Formally specified as: $\mathrm{AG}[e_2 \Rightarrow \mathrm{AG}{\sim}e_1]$

- Existence Dependency: $e_1 \rightarrow e_2$.
  If event e_1 occurs sometimes, then event e_2 also occurs sometimes.
  Alternatively, there is no computation such that e_2 does not occur until a state *s* is reached where *s* satisfies [e_1 is executed in *s*, and subsequently, e_2 never occurs].
  Formally specified as: ${\sim}\mathrm{E}[{\sim}e_2 \,\mathrm{U}\, (e_1 \wedge \mathrm{EG}{\sim}e_2)$

- Conditional Existence Dependency [KL91]: $e_1 \rightarrow (e_2 \rightarrow e_3)$]

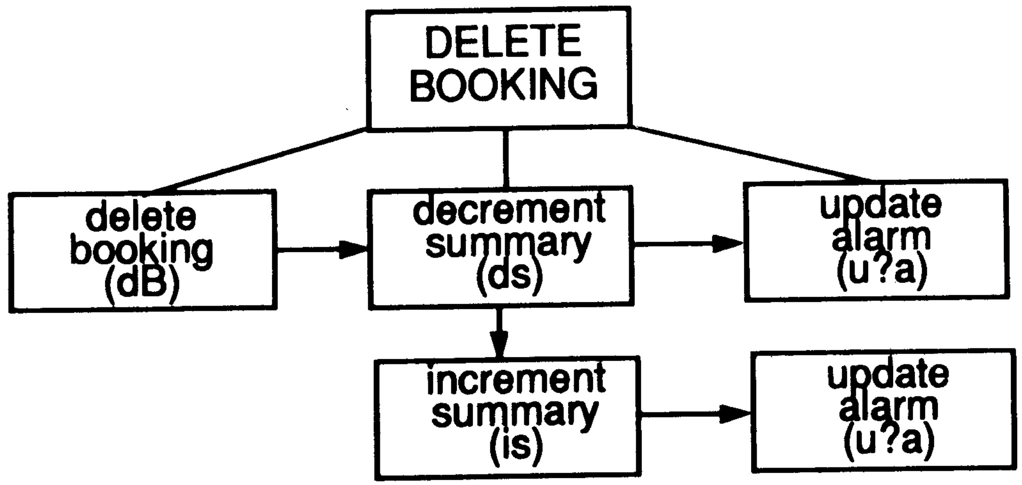
Examples from multidatabase transaction models:

- Commit Dependency [CR92]: $cm_B < cm_A$
- Abort Dependency [CR92]: $ab_B \rightarrow ab_A$

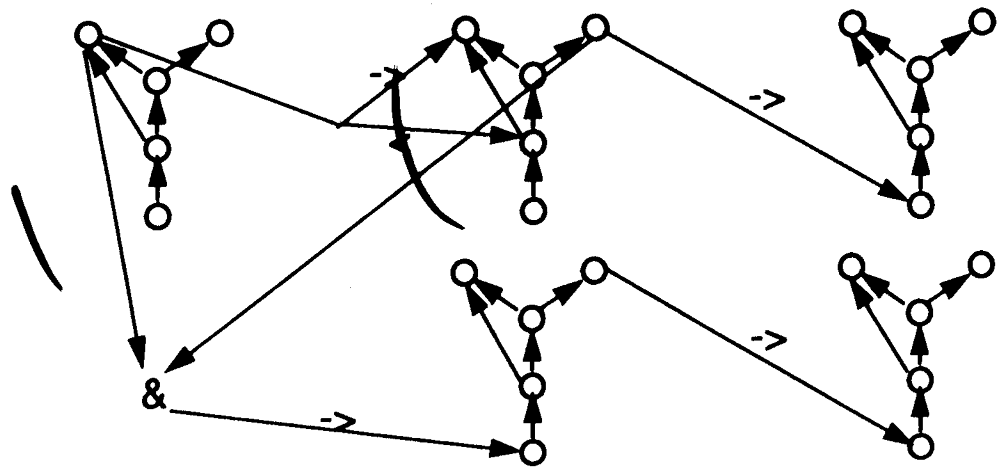
[Attie et al 93]]

An Example

Task Graph



Intertask Dependencies



s(dB) -> s(dS)
c(ds) -> s(u?a)
c(iS) -> s(u?a)
(a(dB) & c(dS)) -> s(iS)
(a(dB) < d(dS)) -> a(dS)

[Woelk et al 93]

Enforceable Dependencies

- Dependencies may not be enforceable.

  For example, *ab(A) -> cm(B)*

- Event attributes determine whether a dependency is enforceable. For example,
  - $e_1 \rightarrow e_2$ is run-time enforceable if

    rejectable(e_1) [delay e_1 until e_2 is submitted, reject e_1 if task 2 terminated without submitting e_2],

    or forcible(e_2) [force execution of e_2 when e_1 is accepted for execution].

    after

  - $e_1 < e_2$ is run-time enforceable if

    rejectable(e_1) [let e_2 be executed when it is submitted, thereafter reject e_1 if submitted],

    or delayable(e_2) [delay e_2 until either e_1 has been accepted for execution, or task 1 has terminated without issuing e_1].

[Attie et al 93]

Using CTL for dependency specification

CTL is Computational Tree Logic [Emerson 90].

- formal semantics
  (propositional branching-time temporal logic: propositional logic and temporal operators)
- expressive, e.g., nesting of dependencies
- tools/algorithms for consistency and completeness checking
- limited real-time extension:
  dependencies that involve absolute clocks or relative-time service alarms (number of ticks), e.g., express: "$e_1 <^t e_2$ such that e_2 occurs within t time units of e_1" or "$e_1 \to^t e_2$ such that e_2 occurs no later than t time units after e_1"
- algorithms for automatic synthesis of automata for reactive systems
  (to develop scheduler to enforce intertask dependencies)

[Attie et al 93]]

Dependency Automata

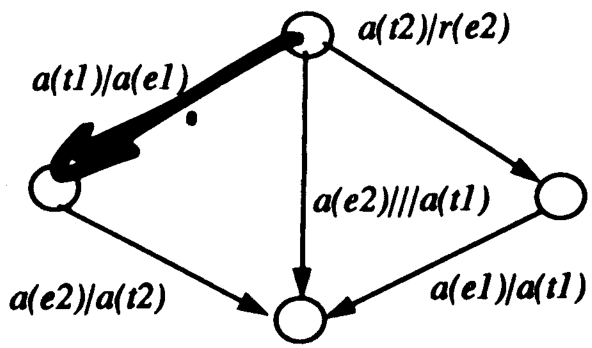
For a dependency $D(e_1,\ldots,e_k)$, create a FSM A for enforcing D.

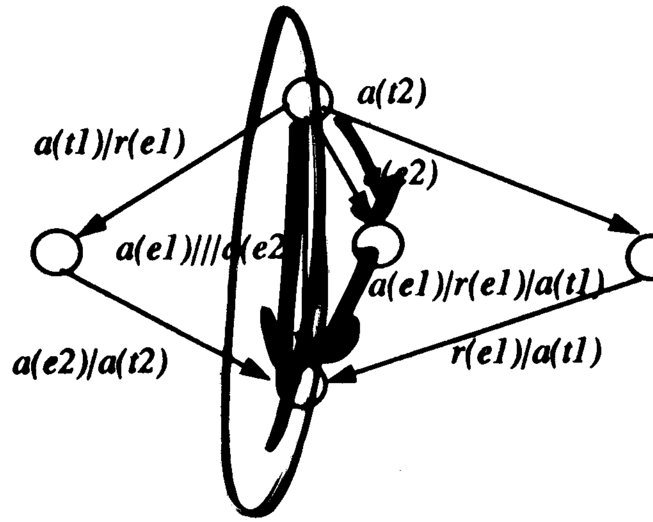
- Automaton A represents D for internal processing. Each path in A denotes a set of computations on which D is satisfied.

- A can be synthesized *automatically* from
  - the CTL formula for D, and
  - the attributes of the events $e_1,\ldots,e_k$

[Attie et al 93]]

Example Dependency Automata





$e_1 < e_2$;
rejectable(e_2) and delayable (e_2)

e_1 -> e_2;
rejectable (e_1) and delayable (e_2)

[Attie et al 93]]

Beyond Dependencies -- Task Coordination Requirements

Statically -- a precondition for starting a task or initiating a transition in a task.

Preconditions may be specified with dependencies involving:

- execution states of other tasks
- output values of other tasks
- external variables (events outside the workflow, time,..)

E.g., execution dependencies, data/value dependencies, temporal dependencies in Flexible Transactions [Elmagarmid et al 90], ConTracts [Reuter 89], Multitransactions [Garcia-Molina et al 90], Multidatabase Transactions [Rusinkiewicz et al 92]....

Dynamically--

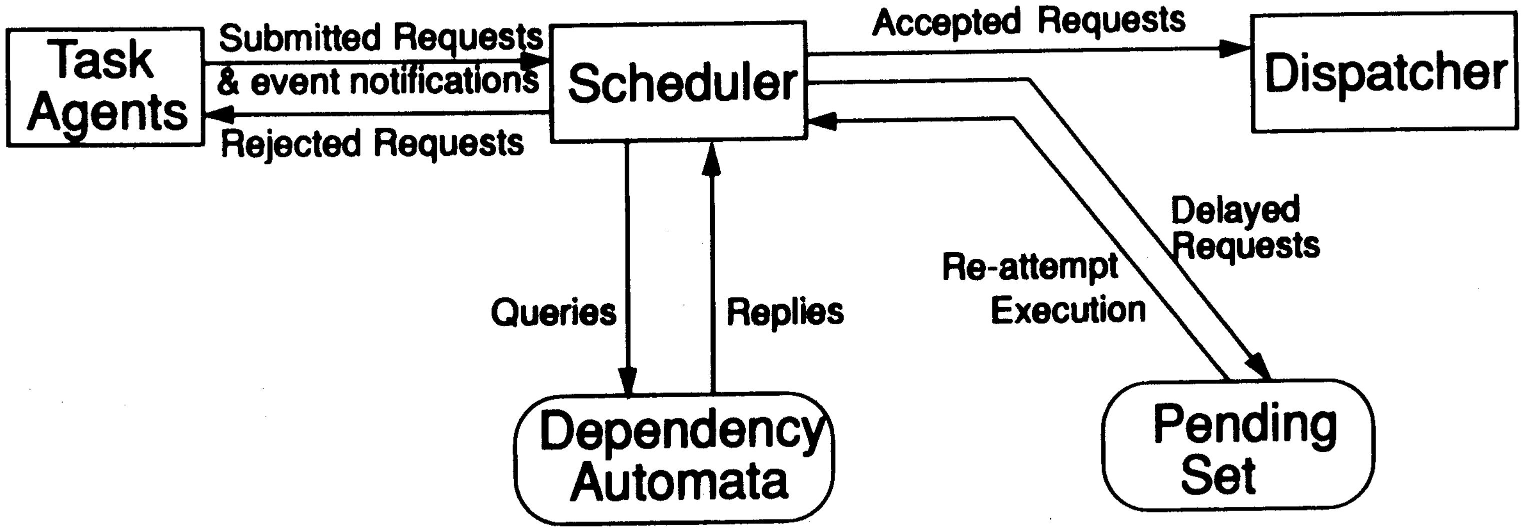
Created when executing a workflow.

Long-running activities [Dayal et al 91], Polytransactions [Rusinkiewicz and Sheth 91].

[Rusinkiewicz and Sheth 93]

Execution Model

(a centralized approach)



[Attie et al 93]

Enforcing Multiple Dependencies

Pathset: one path corresponding to ε from each relevant dependency automaton.

A *desired pathset* must:

- accept ε,
- begin in the current global state of the scheduler,
- be order-consistent,
  - different paths in the set must agree on the order of execution of each pair of events
- be a-closed or r-closed, and
  - for any event that is accepted or rejected, paths from each automaton referring to that event must be included and must agree on whether to accept it or reject it
- be executable
  - all rejected events must have been submitted and all accepted events must have been submitted or be forcible.

[Attie et al 93]

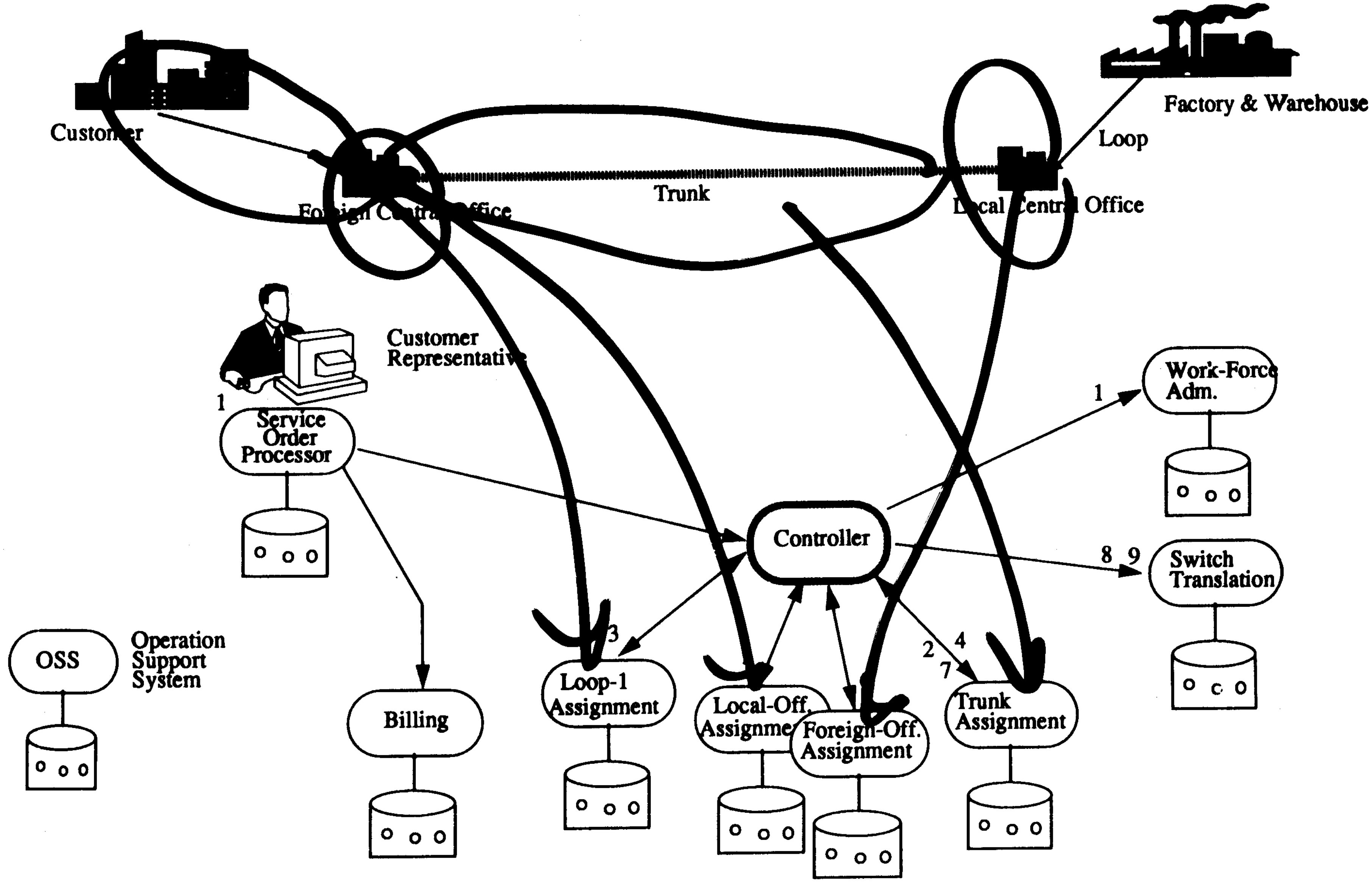
Scheduler Operation (An Example)

Consider only $e_1 < e_2$ and $e_1 \rightarrow e_2$ dependencies, where both e_1 and e_2 are rejectable (e.g., all dependencies for SAGAs can be expressed using these). Corresponding automata $A_<$ and $A_{\rightarrow}$.

- e_1 is submitted.
  - $a(e_1)$ in $A_<$. No path in $A_{\rightarrow}$ with e_1. e_1 added to pending set.
- e_2 is submitted.
  - $A_{\rightarrow}$: $a(e_2);a(e_1)$ and $a(e_2)|||a(e_1)$.
  - *a-closure* forces searching $A_<$ for a path that accepts both e_1 and e_2. Only such path is $a(e_1);a(e_2)$ which is not order-consistent with $a(e_2);a(e_1)$.
  - Viable pathset is $\{a(e_1);a(e_2),\ a(e_2)|||a(e_1)\}$.
  - Partial order consistent with this is e_1 and then e_2.

[Attie et al 93]

An Example Workflow in a Multisystem Application: Provisioning a Telecommunication Service



About the environment

- multiple existing heterogeneous “closed” application systems
  - each system developed independently to automate a business function
  - each with own databases multiple existing heterogeneous “closed” application systems
  - each has predefined interface (“contracts”)
- multisystem application implemented using <u>dedicated</u> controller-
  - hard-coded, difficult to change work-flow
  - use of queued message paradigm
  - no use of transaction paradigm for multisystem application
    - – application specific and application managed concurrency control and recovery

[Ansari et al 92]

ted Work, Work in Progress and Future Work

antiating and monitoring of workflows (completed)

e transactional workflow specification and

ıkiewicz/Sheth 93a,b] [Bellcore - UofH]

/Semantic Transaction *vs.* Workflow [Breitbart et al 93]

Control and Recovery that exploit application and

ıtics [Jin et al 93a,b] [UofH - Bellcore]

ɔuted scheduler [one scheduler per workflow] (Jin et

H]

:heduler [MCC]

ifications of individual tasks (messages to cation Application Systems-OSSs) [Bellcore]

cification and testing of workflows

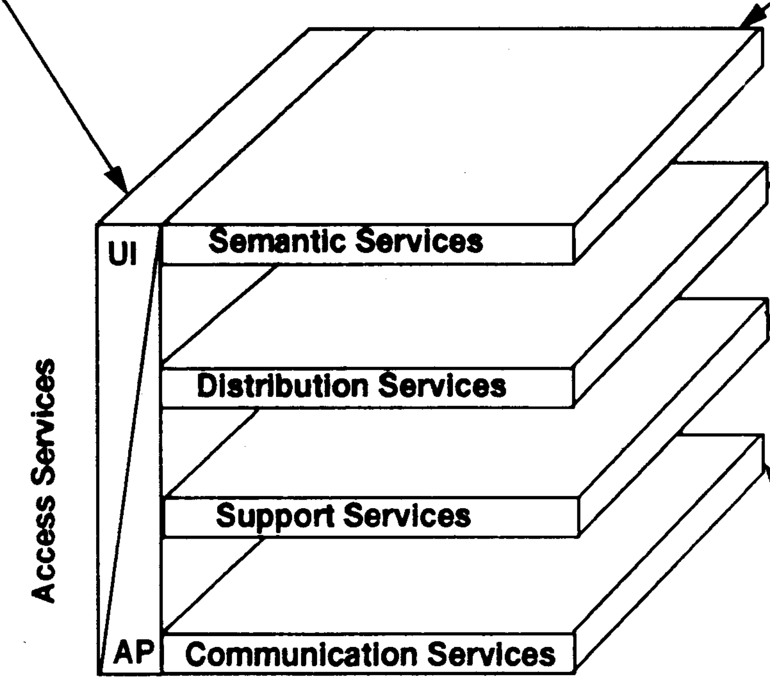
Conclusions

Formal approach to specifying and executing (aspects of) workflows.

- Specification:
  - task skeletons
  - significant event attributes
  - intertask dependencies
- Execution
  - executable/postulative specification
    - correct and safe execution
  - one approach to scheduling- implemented
    - centralized, high computational cost, usable in some practical environments
- More needs to be done, in progress

Carnot Architecture

Access Services -
- 2D & 3D Graphical Interaction Environment
- Deductive Computing
- Application Frameworks



Semantic Services -
- Enterprise Modeling and Model Integration
- Knowledge Discovery
- Application Dredging

Distribution Services -
- Relaxed Transaction Processing
- Communicating Agents
- Concept-Based Security
- Work Flow Manager
- Declarative Resource Constraint Base
- Legacy System Access (ADDS)

Support Services -
•Extensible Services Switch •RDA •TP •IRDS •ORB
•X.500 •X.400 •Security •SNMP •CMIP •EDI

Communication Services -
•OSI •Internet ~~•X.25~~ •SNA •DCE •Atlas
•SMDS •Frame Relay •FDDI •BISDN