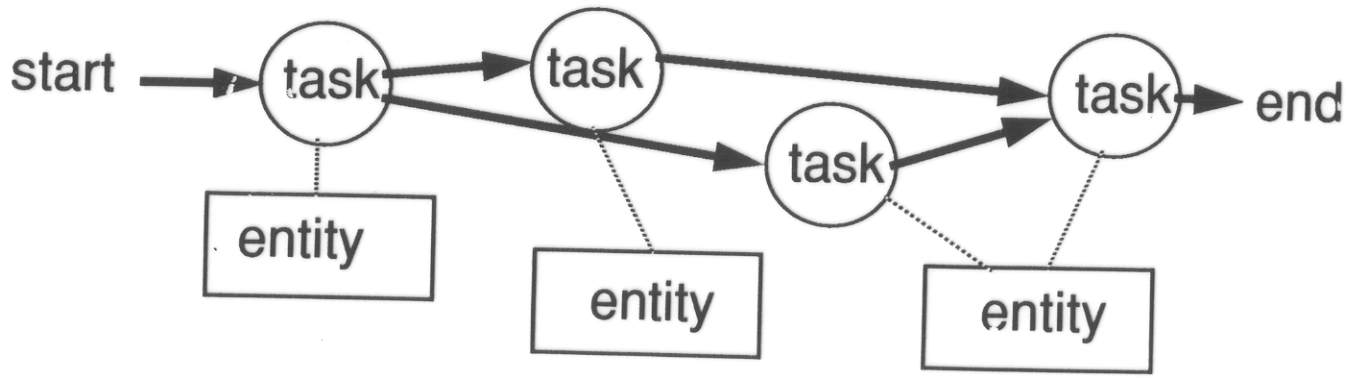
Talk Outline

- Introduction to the basic concepts of workflow
- Formal model for specification and scheduling workflows
- Prototype and applications



Transactional Workflow Management

Three Components:

- **Specification:**

  declarative, flexible specification of tasks/and dependencies

  extended transaction models [Elmagarmid 92] (e.g., Flexible Transaction [Elmagarmid/Rusinkiewicz et al. 90]; ACTA [Ramamritham/ Chrysanthis 91/92], dependency specifications [Klein 91]

- **Scheduling:**

  efficient, maximal parallelism, exploit task and system semantics

  e.g., (L.0) [Cameron et al. 91]; VPL [Kuehn et al 92]

- **Executing:**

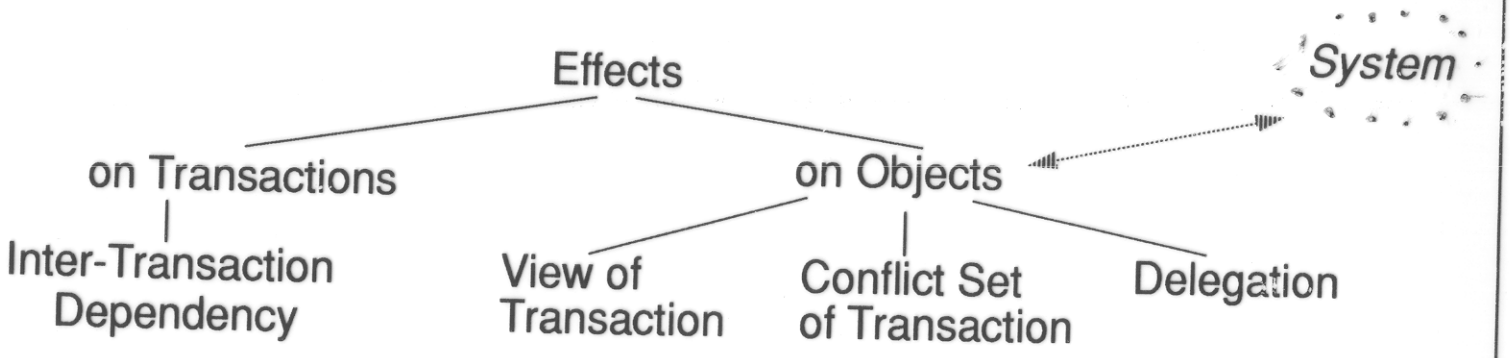
  manage execution of tasks/transactions on heterogeneous, autonomous component systems

  e.g., DOL System, Narada [Rusinkiewicz/UofH], Interbase [Elmagarmid/Purdue]

The ACTA Framework

Specification of the effects of transactions on other transactions and their effect on objects.

- significant events associated with transactions
- histories and conditions on event occurrences
- objects, operations, and conflicts



[Chrysanthis & Ramamritham 91/92, Ramamritham & Chrysanthis 92]

Uses of ACTA

- understand transactions properties
- compare transaction models

Not sufficient to generate schedule, or to support multiple transaction model interoperability.

Beyond ACTA

- "postulative specifications" [Rusinkiewicz]
- "transaction specification toolkit"/ a-la-carte transaction
- specification of execution semantics [Sheth and Kalinichenko]

Workflow Specification Model

- Significant events [$e, e_i, e_j, ..$]:

  events from tasks submitted by users/system (user events/ system events); associate with a task ~~agent~~ skeletons

- Task Skeletons:

  an abstract representation of the actual task that hides irrelevant details of sequential computation of the task

- Intertask dependencies [$D(e_1,...,e_n)$]:

  constraints on occurrence and temporal order of events

- System & application semantics

Workflow Specification Model

- "Significant events" [$e, e_i, e_j, ..$]:

  events from tasks submitted by users/system (user events/ system events); associate with a task agent

- Task Skeletons:

  an abstract representation of the actual task that hides irrelevant details of sequential computation of the task

- Intertask dependencies [$D(e_1,...,e_n)$]:

  constraints on occurrence and temporal order of events

Significant Events

Significant event types for database applications: *st, ab, pr, cm*

Possible attributes of an event type:

- Forcible: the system can always force the execution
- Rejectable: the system can always reject the event
- Delayable: the system can delay execution of the event
  (every non-real-time significant events are delayable)

User Tasks	Forcible	Rejectable
cm	×	✓
ab	×	×
pr	×	×
st	×	✓

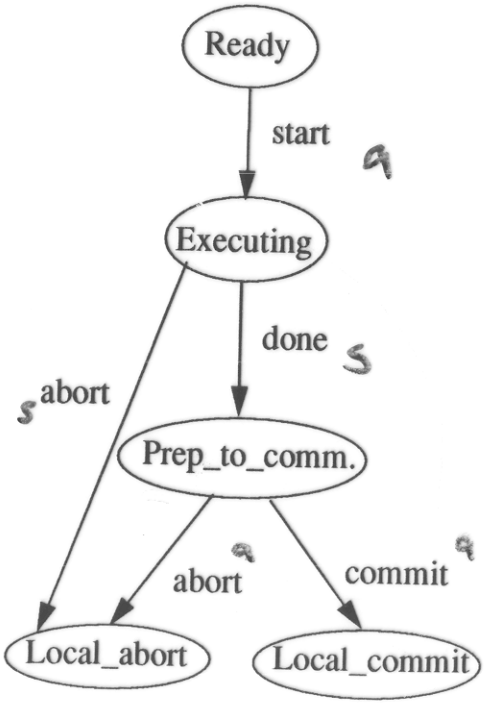
System Tasks	Forcible	Rejectable
cm	×	✓
ab	✓	×
pr	×	×
st	✓	✓

Table 1: Attribute Tables for Significant Events

Task Skeleton

- different skeleton depending on application and the system
  - different states (e.g., no precommit)
  - different significant events submitted by application/user and system

Example:



Intertask Dependencies

Klein's primitives [KL91]:

- Order Dependency: $e_1 < e_2$.
  If both e1 and e2 occur, then e1 precedes e2.
  Alternatively, in CTL: if e_2 occurs, e_1 cannot occur subsequently.
  Formally specified as: AG[e_2 => AG~e_1]
- Existence Dependency: e_1 -> 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: ~E[~e_2 U (e_1 ∧ EG~e_2)
- Conditional Existence Dependency [KL91]: e_1 -> (e_2 -> e_3)]

Examples from multidatabase transaction models:

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

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 -> e_2 is run-time enforceable if

    rejectable(e_1) [delay e_1^1 until e_2^2 is submitted, reject $e1$ if task 2 terminated without submitting e_2^2],

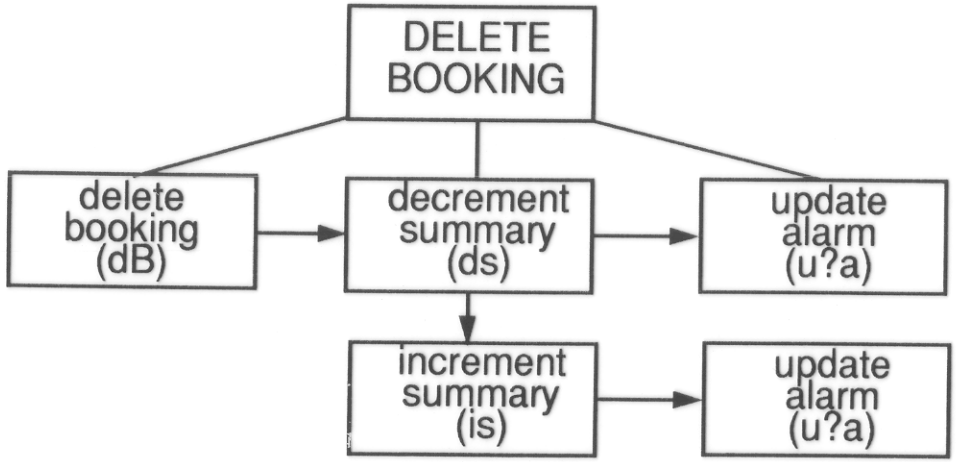
    or forcible(e_2) [force execution of e_2 when e_1 is accepted for execution].
  - $e_1 < e_2$ is run-time enforceable if

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

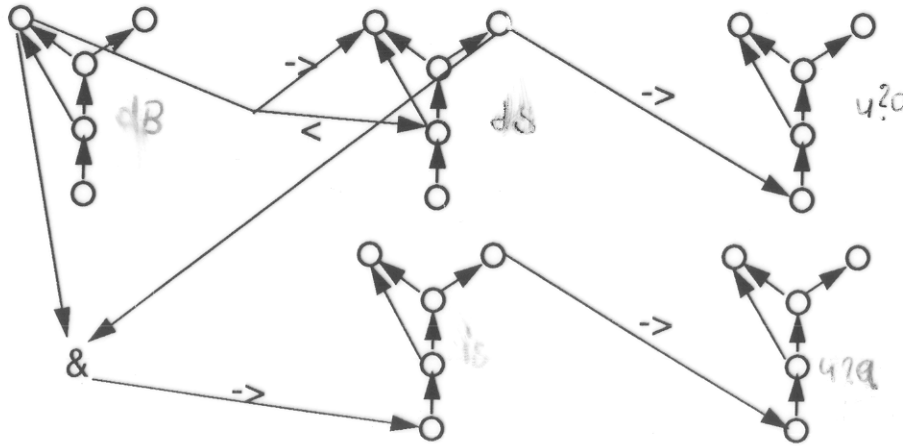
    or delayable(e_2) [delay e_2 until either e_1 has been accepted for execution, or task 1 has terminated without issuing e_1].

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)

Using CTL for dependency specification

- 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 temporal extension:
  dependencies that involve absolute clocks or relative-time service alarms (number of ticks), e.g., express: "$e_1 < e_2$ such that e_2 occurs within t time units of e_1" or "e_1 -> 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)

Using CTL for dependency specification

- 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 temporal extension:
  dependencies that involve absolute clocks or relative-time service alarms (number of ticks), e.g., express: "$e_1 < e_2$ such that e_2 occurs within t time units of e_1" or "e_1 -> 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)

Dependency Automata

ıcy D(e_1,...,e_k), create a FSM A for enforcing D.
nthesized automatically from
formula for D, and
ɔutes of the events e_1,...,e_k
<S,s', Σ, ρ>, where
ɛt of states, s' is the initial state, Σ is the alphabet,
the transition relation.
ɛs of Σ are of the form
$_m$), i.e., A accepts ε_1 through ε_m
her relevant significant event e_1 or its termination

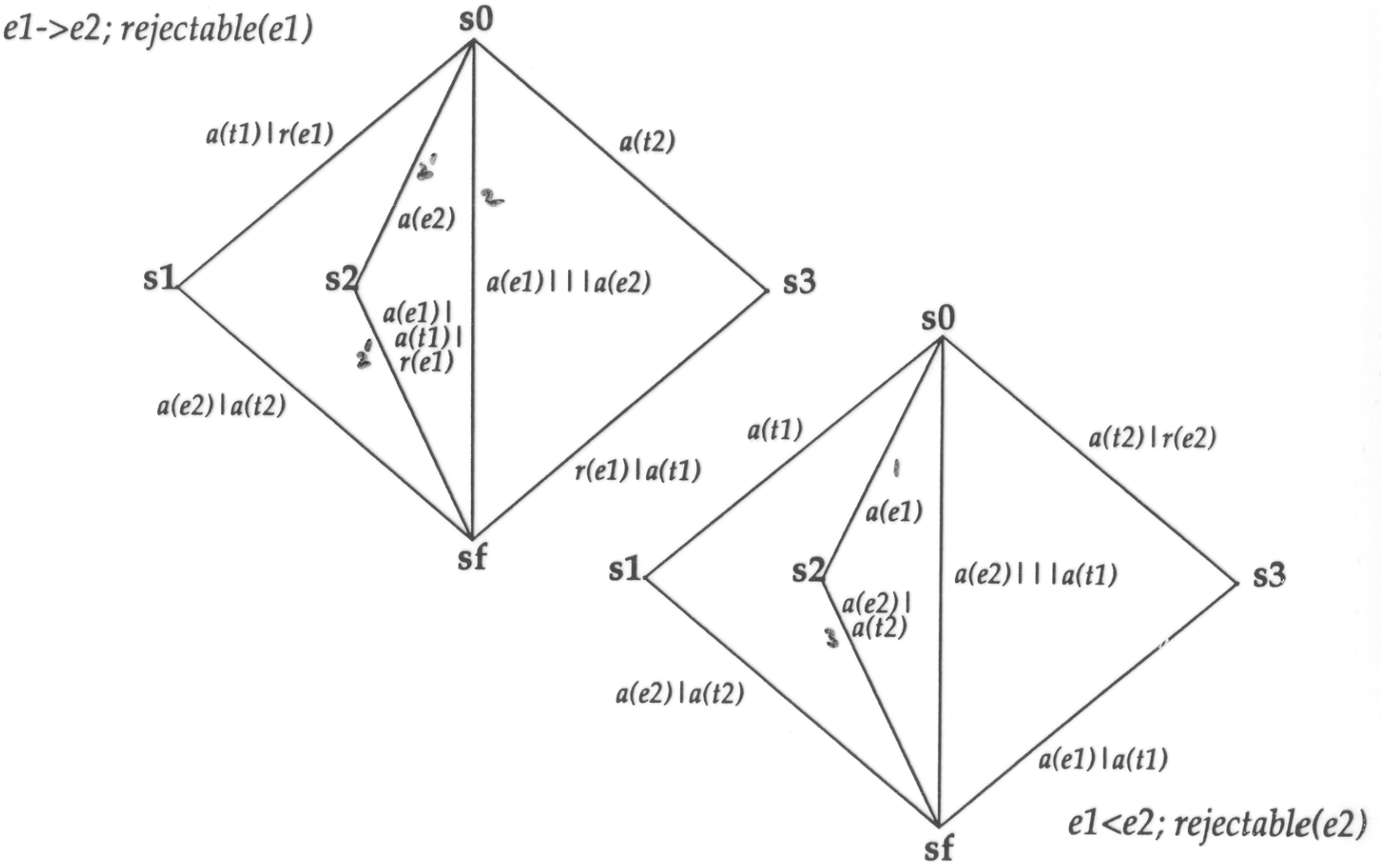
$_m$), i.e., A rejects e_1 through e_m,
σ_n, i.e., interleaving of accept operations in σ_1
σ_n, and
, i.e., accept op. in σ_i may occur before accept op. in

Dependency Automata

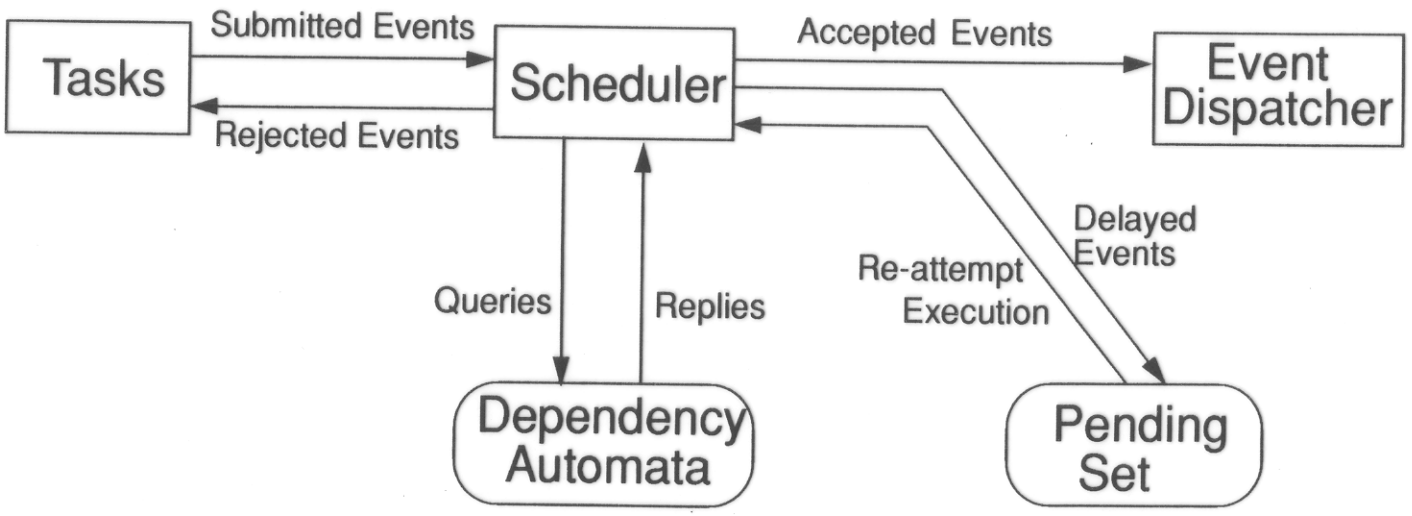
For a dependency $D(e_1,...,e_k)$, create a FSM A for enforcing D.

- A can be synthesized automatically from
  - the CTL formula for D, and
  - the attributes of the events $e_1,...,e_k$
- A is a tuple $<S,s', \Sigma, \rho>$, where
  - S is a set of states, s' is the initial state, Σ is the alphabet, and ρ is the transition relation.
  - Elements of Σ are of the form
    $a(\varepsilon_1,...,\varepsilon_m)$, i.e., A accepts ε_1 through ε_m
    [ε_1 is either relevant significant event e_1 or its termination event t_1,
    $r(e_1,...,e_m)$, i.e., A rejects e_1 through e_m,
    $\sigma_1|||...|||\sigma_n$, i.e., interleaving of accept operations in σ_1 through σ_n, and
    $\sigma_1;...;\sigma_n$, i.e., accept op. in σ_i may occur before accept op. in σ_{i+1}

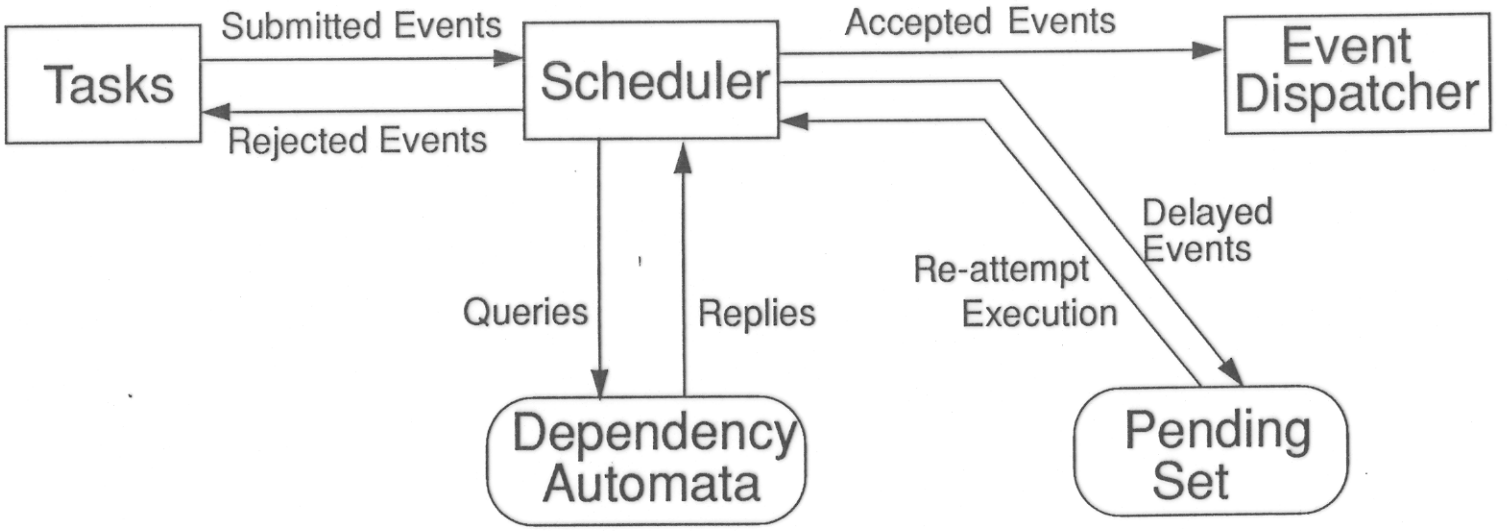
Example Automata



Execution Model



Execution Model



forcing Multiple Depenencies

ıe path corresponding to ε fromach relevant
;y automaton.

ɔathset must:

ε,

ı the current global state of the:heduler,

ɛr-consistent,

ɛnt paths in the set must agree on the ordef execution of each pari of

ɔsed or r-closed, and

y event that is accepted or rejected, path fr(each automaton referring
ɛnt must be included and must agree on wlher to accept it or reject it

ɔutable

ɛcted events must have been submitted anll accepted events must
ı submitted or be forcible.

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,

  i.e., different paths in the set must agree on the order of execution of each pari of events
- be a-closed or r-closed, and

  i.e., for any event that is accepted or rejected, path from each automaton referring to that event must be included and must agree on whether to accept it or reject it
- be executable

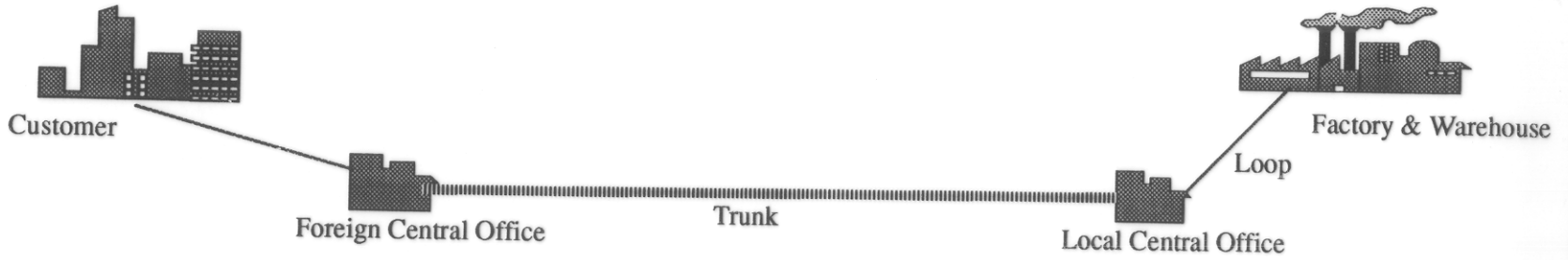
  i.e., all rejected events must have been submitted and all accepted events must have been submitted or be forcible.

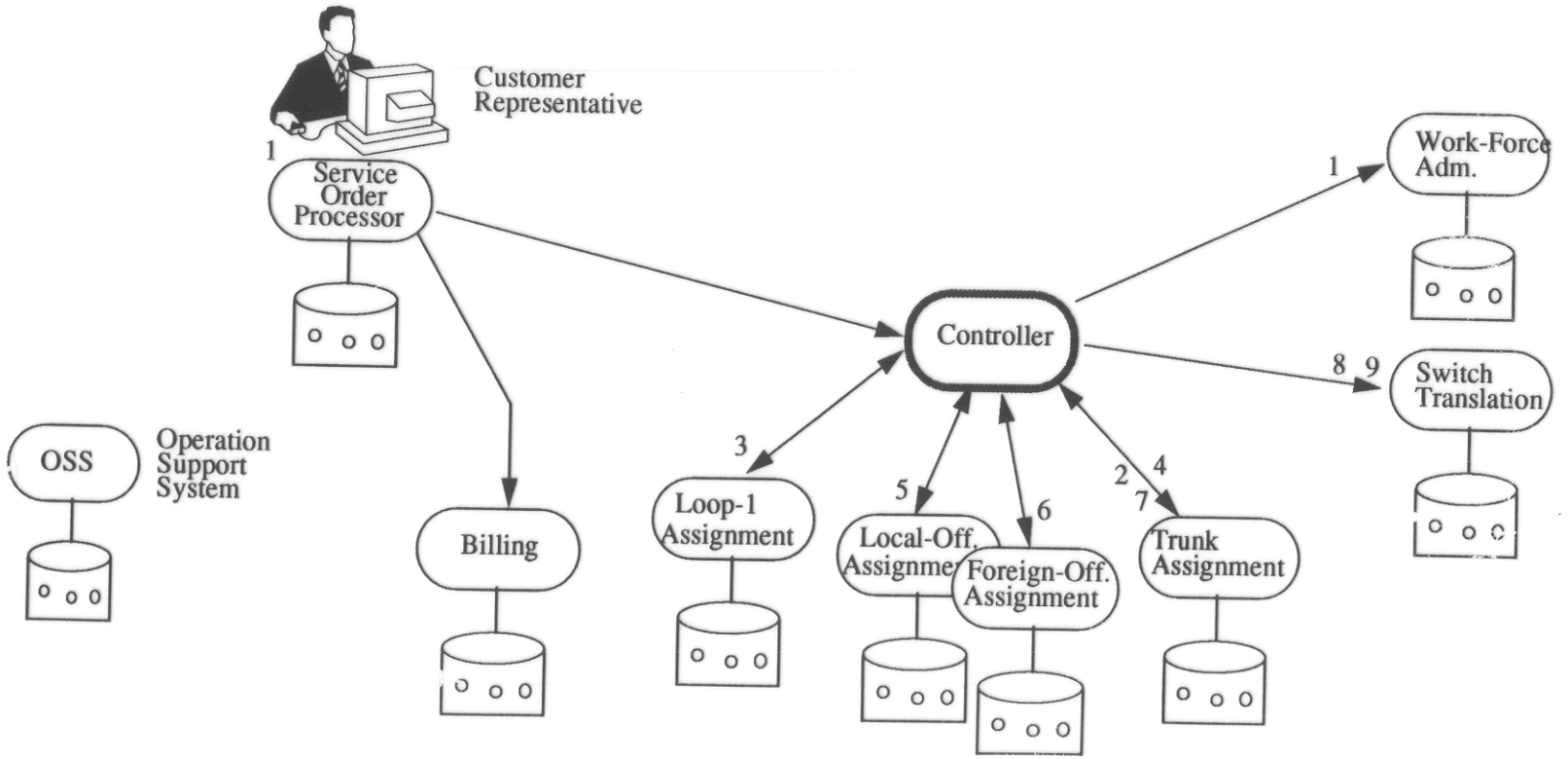
Scheduler Operation (An Example)

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 $A_{->}$.

- e_1 is submitted.
  - $a(e_1)$ in $A_<$. No path in $A_{->}$ with e_1. e_1 added to pending set.
- e2 is submitted.
  - A->: $a(e_2);a(e_1)$ and $a(e_2)|||a(e_1)$.
  - *a-closur*e forces searching A< for a path that accepts both e1 and e2. 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.

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



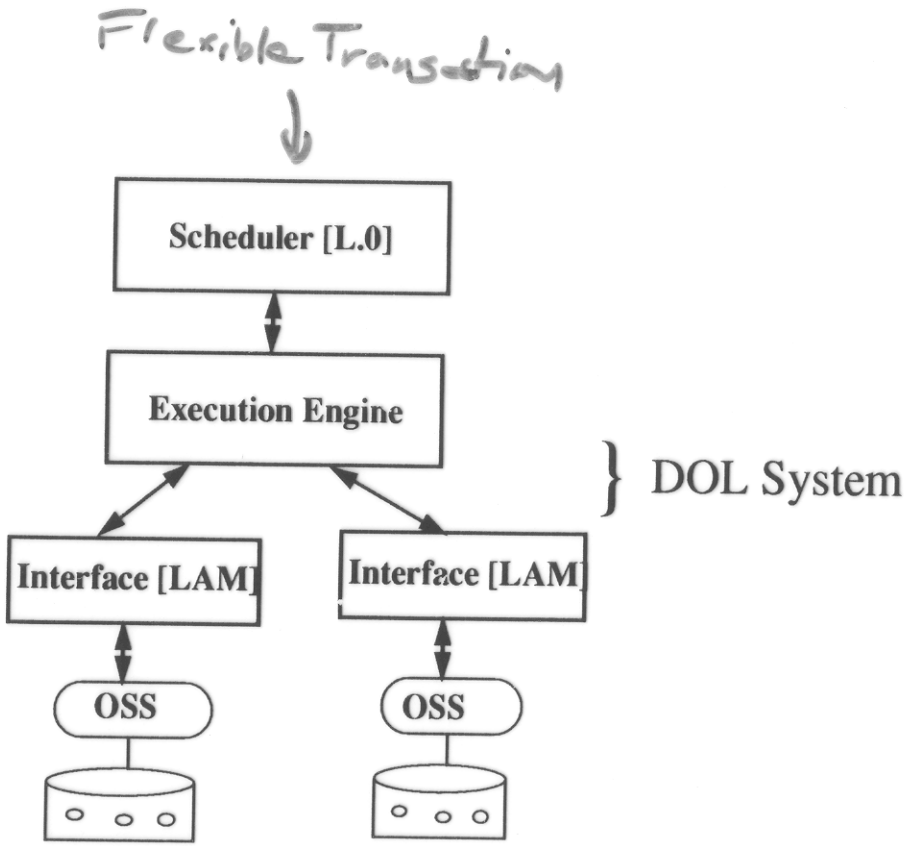


About the environment

- 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 dedicated 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

PROMT

(A system for Processing Multidatabase Transactions)



OSSs (application systems) emulated using DBMSs.

[Bellcore, UofH]

Beyond PROMT

- Scheduler based on a formal framework
- Concurrency Control and Recovery that exploit application and system semantics (implementation in progress)
- Interface with real OSSs
- Graphical specification and testing of workflows (in progress)
- Graphical instantiation and monitoring of workflows
- Detailed specifications of individual tasks (messages to OSSs)
- Data dependencies and translations using tables, rules, constants, and syntactic changes

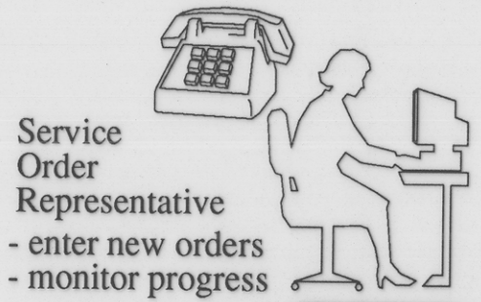
Conclusions

- Gained detailed understanding on issues of transaction workflows.
- Studied and demonstrated applicability of relaxed multidatabase transaction.
- **Developed formal approach to specifying and executing workflows.**
- Completed large-scale prototype.
  Demonstrated with a real application.
  Deployment considered.

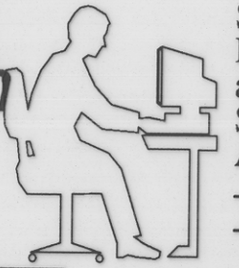
PROMT Conclusions

- Flexible Transaction model seems natural to model multi-system work-flows
  - Some extensions/improvements performed/desired:
    - specification of acceptable states
    - additional execution dependencies
- Use of relaxed transaction technology appropriate
  - already practiced informally, but relaxed transactions provide better and precise understanding of relaxation of ACID properties
  - separation of control from individual (sub) transactions
  - functionally better than current practice of message queues or store-and-forward with manual intervention
  - better modeling of component systems (subtransaction state transitions)
- Yet to understand how to handle manual interventions

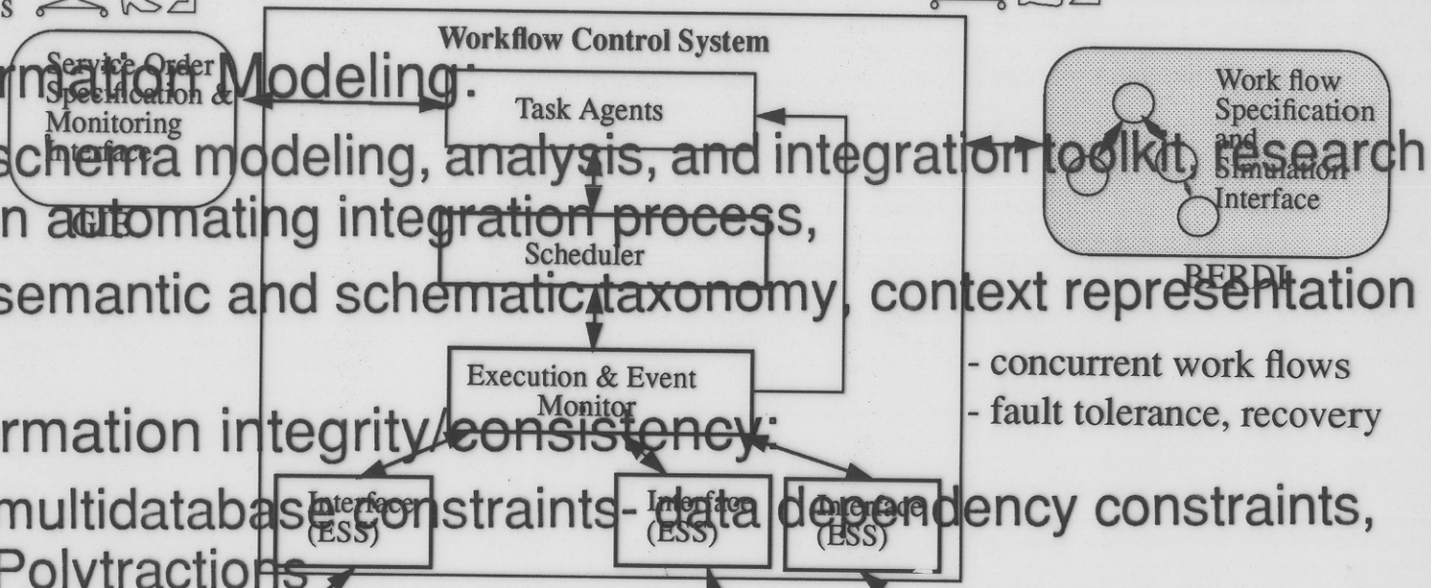
Prototype 2: SPIRIT
Recent Research



Service Order Representative
- enter new orders
- monitor progress

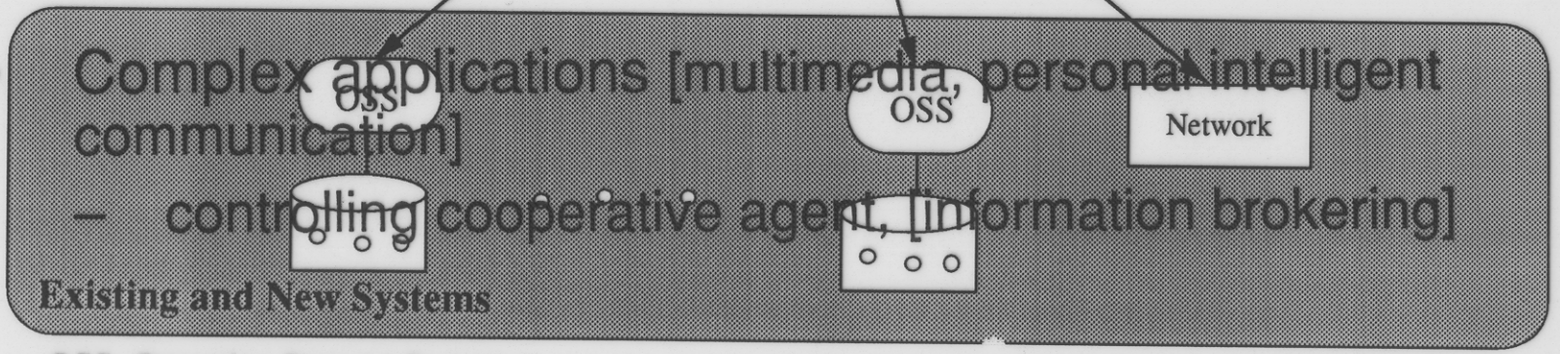


Service Developer and System Analyst
- specify new services
- test/simulate



- Information Modeling:
  - schema modeling, analysis, and integration toolkit, research in automating integration process,
  - semantic and schematic taxonomy, context representation
- Information integrity/consistency:
  - multidatabase constraints- data dependency constraints, Polytractions
- Complex applications [multimedia, personal, intelligent communication]
  - controlling cooperative agent, [information brokering]

- concurrent work flows
- fault tolerance, recovery

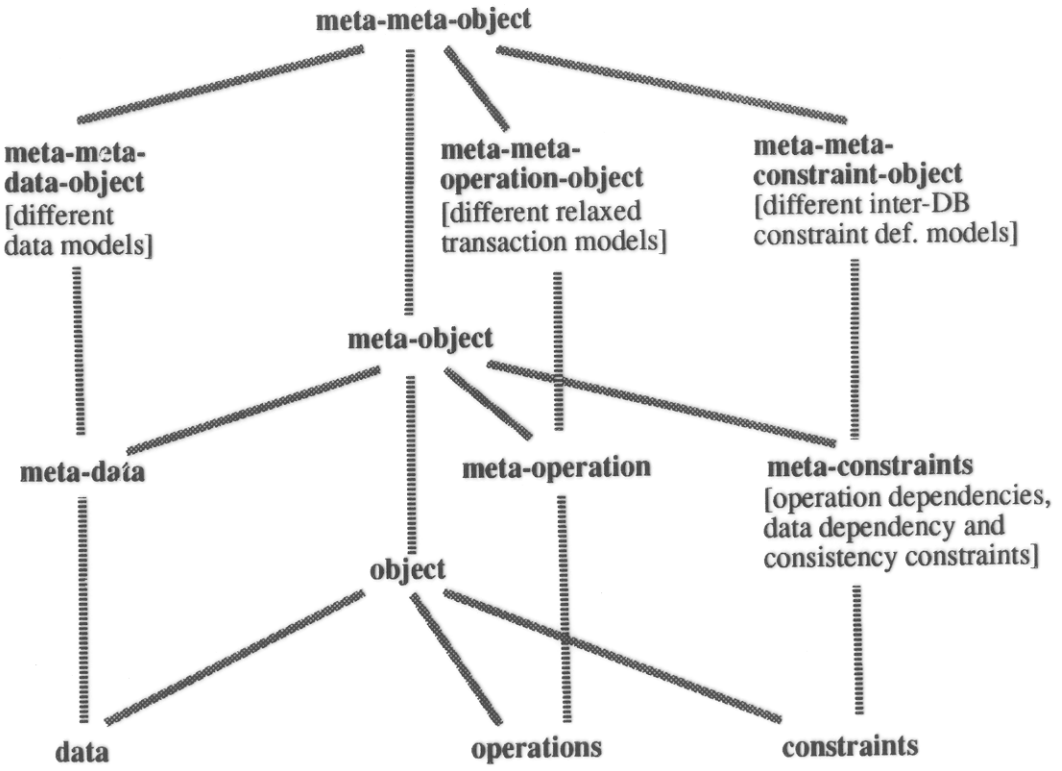


OSS: Operation Support System

[MCC,Bellcore,UofH, Ameritech]

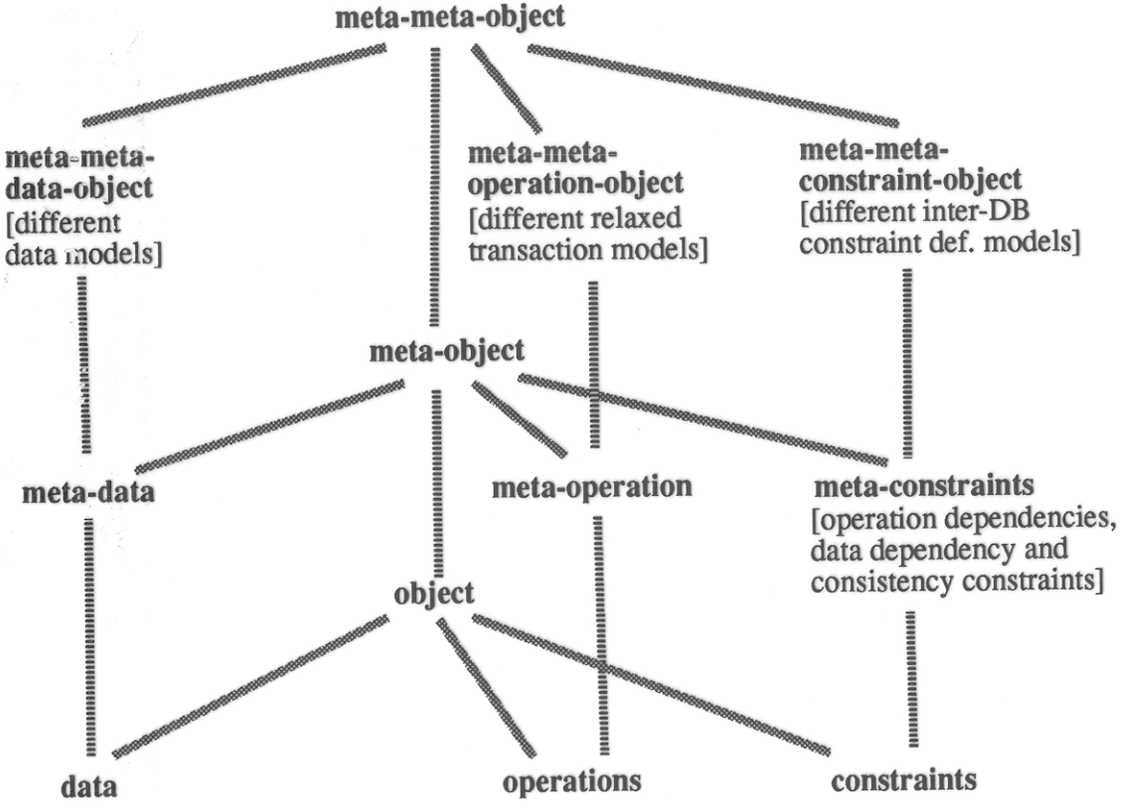
Jan. 1993, Amit P. Sheth

Modeling, Organizing and Managing Information Systems



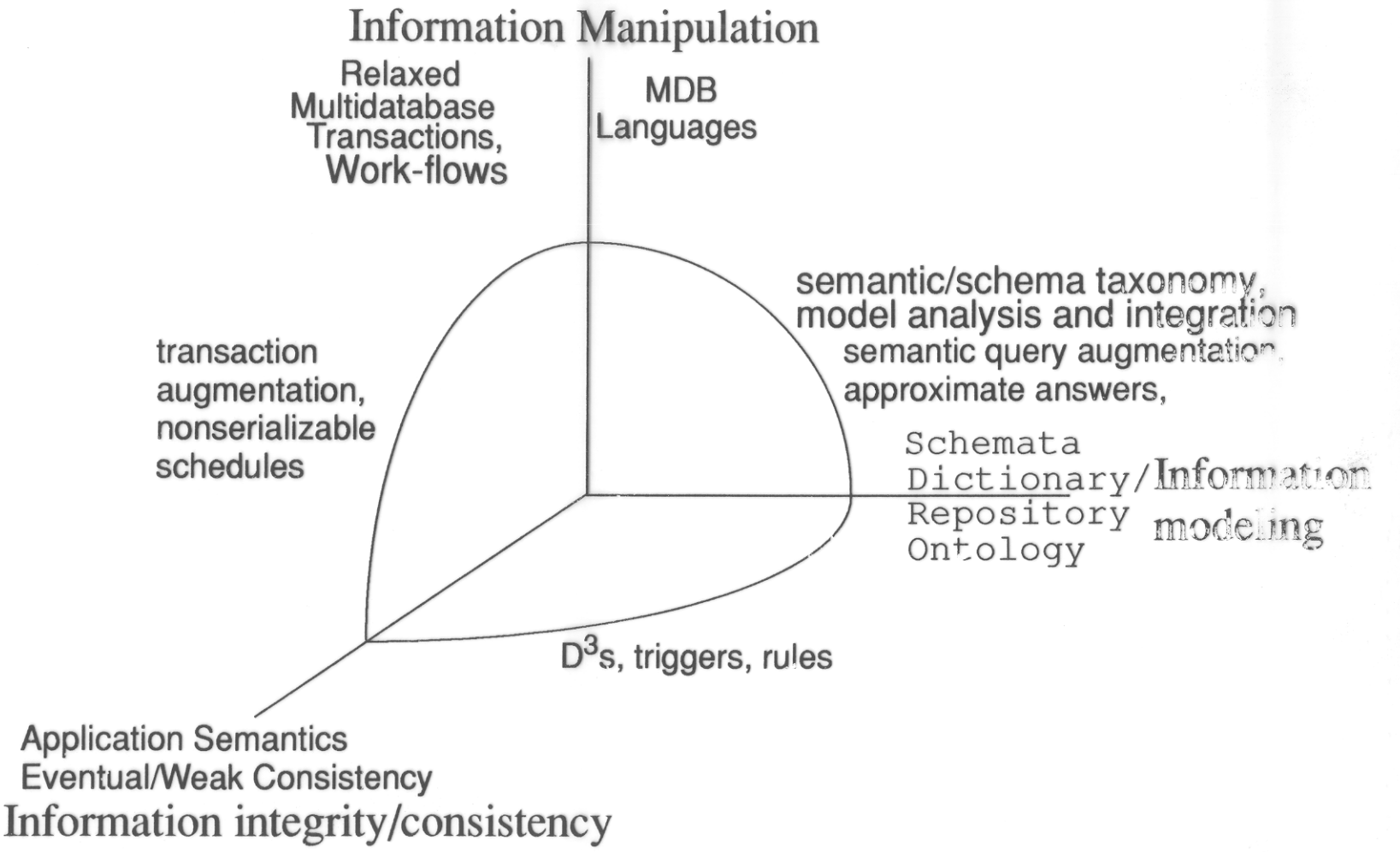
[Sheth, Kalinichenko 92]

Modeling, Organizing and Managing Information Systems



[Sheth, Kalinichenko 92]

Issues in Interoperability in Information Systems



[Sheth and Kalinichenko 92, Sheth92a]