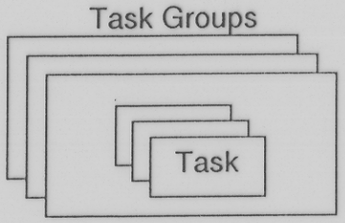
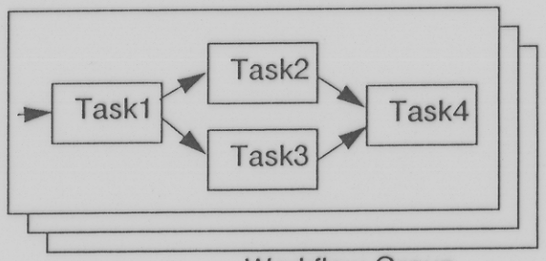
(Distributed) Transaction Processing (DTP) vs Transaction Workflow (TWF)

(Typical Case)

DTP/D-OLTP is focused on efficient execution of relatively simple tasks with no coordination across heterogeneous tasks (in different task groups). Transactional workflows requires coordinated execution of heterogeneous tasks, with varied levels of transactional properties, on a variety of systems.



In DTP



In TWF

- Earlier queued message systems and "chaining of transactions".

  Problems: insufficient control over transaction properties, one type of task, interactions among concurrent activities difficult.

[STDL, Encina,...]

Related Work: Database Literature

- ACID transactions and their nested derivatives

  Problems: inflexible, difficult to implement in multi-systems.
- 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/TSME [Buchmann et al 92, Georgokopoulos et al. 93], Third Generation TP Monitors [Dayal et al. 93], ASSET [Biliris et al. 94]

APRICOT

[Rusinkiewicz and Sheth 94, Mohan tutorial]

Enabling technologies and standards

DCE

CORBA

Notes

X/OPEN TxRPC, ANSA ...

STDL

EDI

....

Workflow Management

Three Components:

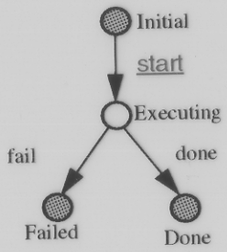
- Specification: Model and Language
  - Task Skeletons/Structures: an abstract representation (state transition diagram) of the actual task that hides irrelevant details of internal computation of the task
  - Significant events or task transition conditions: (requests for) transitions associated with a task agent
  - Intertask dependencies
- Scheduling:
  - safe, correct, optimal/efficient, failure handling; exploit task and system semantics
- Execution:
  - run-time execution of tasks/transactions on heterogeneous, autonomous component systems

Some Technical Challenges

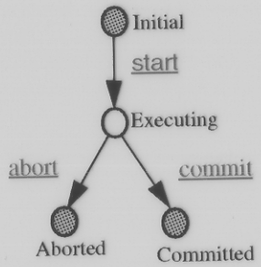
- Different types of tasks => homogeneous modeling; modeling execution behavior
- Correct and Executable (hence formal) specifications and correct and safe execution (including recovery)
- Heterogeneous processing environments and systems => semantics of applications and processing entities, their impact on correct execution and performance
- Performance- many more messages/tasks in "application/ operation automation" as compared to "office/user task automation"

Task Structures

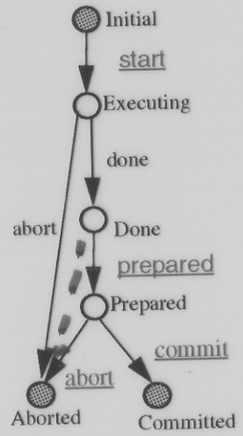
- different state transition diagrams for different types of tasks representing what is *observable* and what is controllable (forcible, rejectable, delayable)



A non-transactional task



A transactional task



An open 2PC transactional task

[Attie et al 93, Rusinkiewicz & Sheth 93, Sheth and Krishnakumar 94]]

Controllable Transitions

Controllable transitions in tasks for database applications: *st, ab, pr, cm*

Possible attributes of a controllable transition:

- 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)

Necessary aspect of specification to support scheduling

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

[Attie et al 93]

Intertask Dependencies

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

Klein's primitives [KL91]:

- Order Dependency: $e_1 < e_2$.
  If both e_1 and e_2 occur, then e_1 precedes e_2.
- Existence Dependency: $e_1 \rightarrow e_2$.
  If event e_1 occurs sometimes, then event e_2 also occurs sometimes.
- Conditional Existence Dependency [KL91]: $e_1 \rightarrow (e_2 \rightarrow e_3)$]

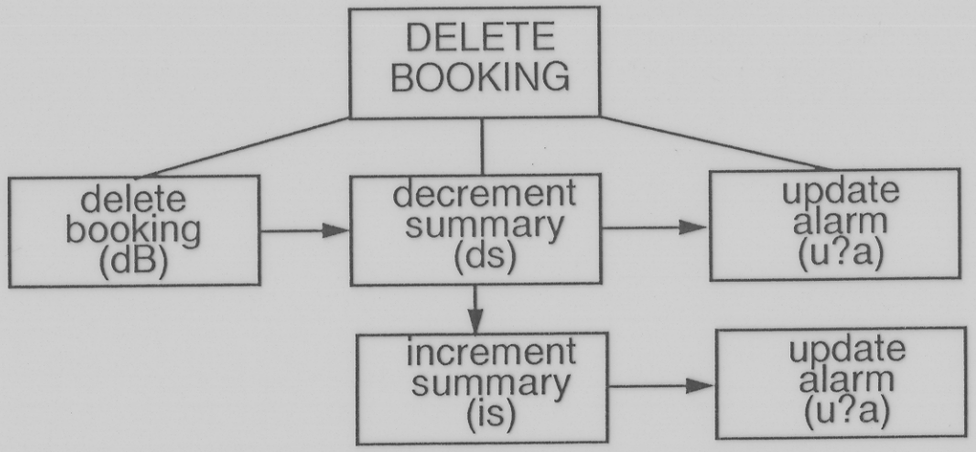
Examples from multidatabase transaction models:

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

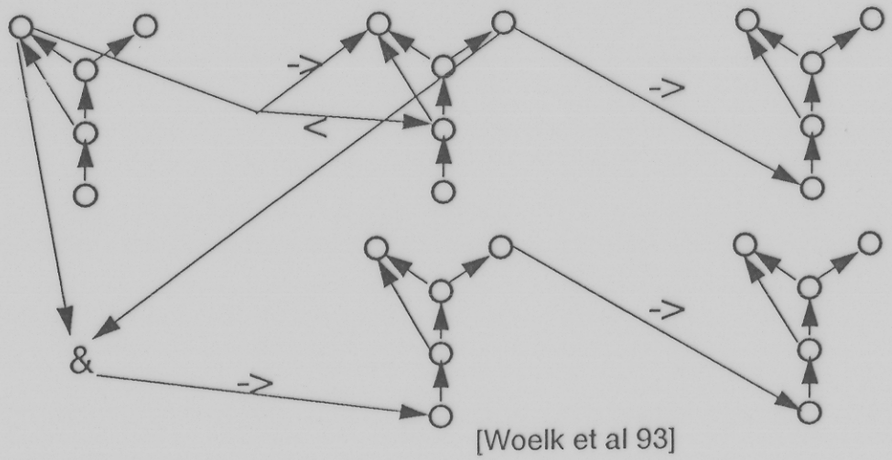
[Attie et al 93, also see ACTA, Bilris et al 94,...]

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]

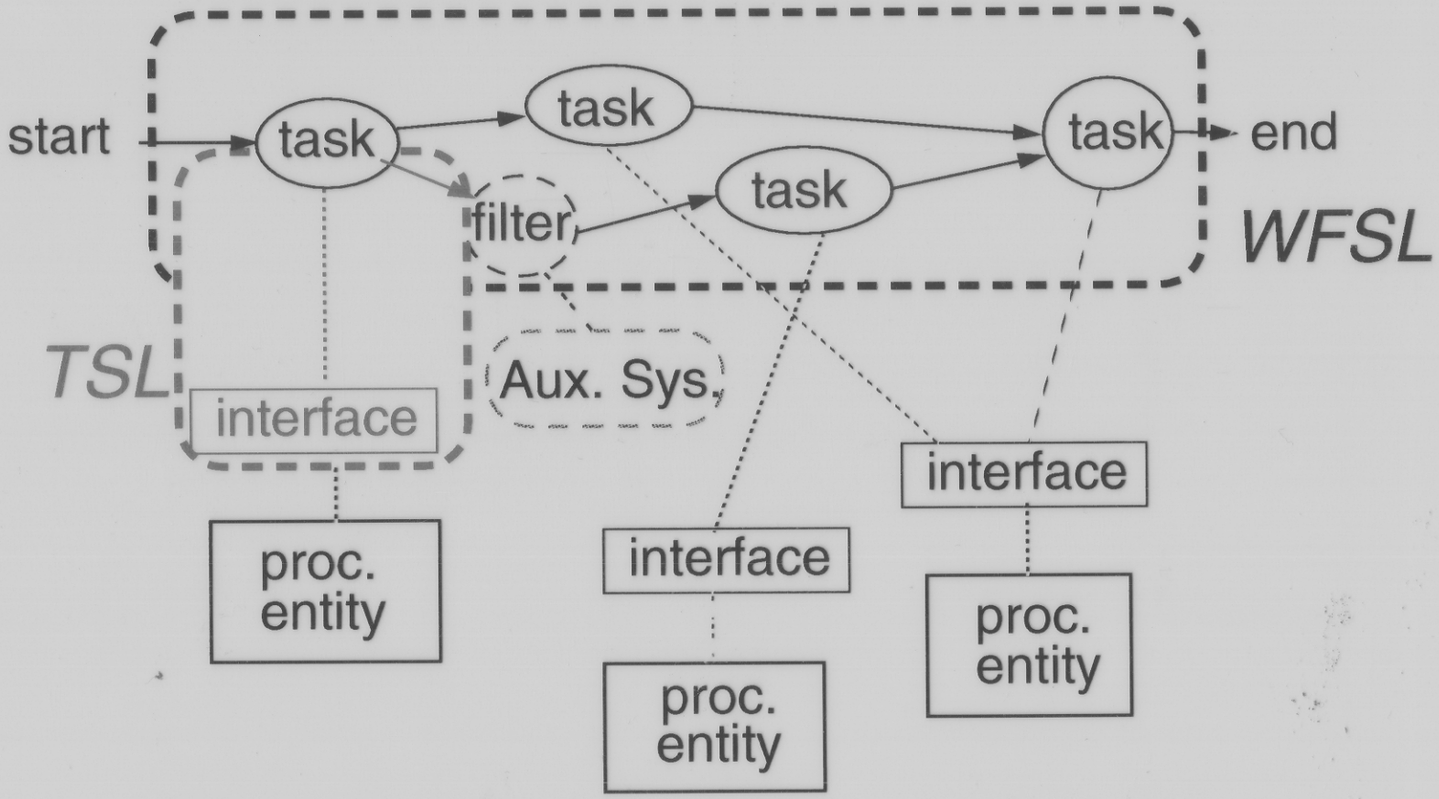
Desirable Specification Language Features

- definition of individual tasks (basic operation definition)
- support for application as well as use tasks (incl. transactions and nonelectronic operations)
- state-based and value based intertask dependencies and data management (control and data flow definitions)
- failure and exception handling
- business rules and constraints
- security and role resolution

[Dayal and Shan 93 (terms in parentheses), Krishnakumar and Sheth 94]

An Approach to Specification

METEOR (Sub-)Languages



WFSL: WorkFlow Specification Language
TSL: Task Specification Language

[Sheth and Krishnakumar 94]

Components of WFSL (partial)

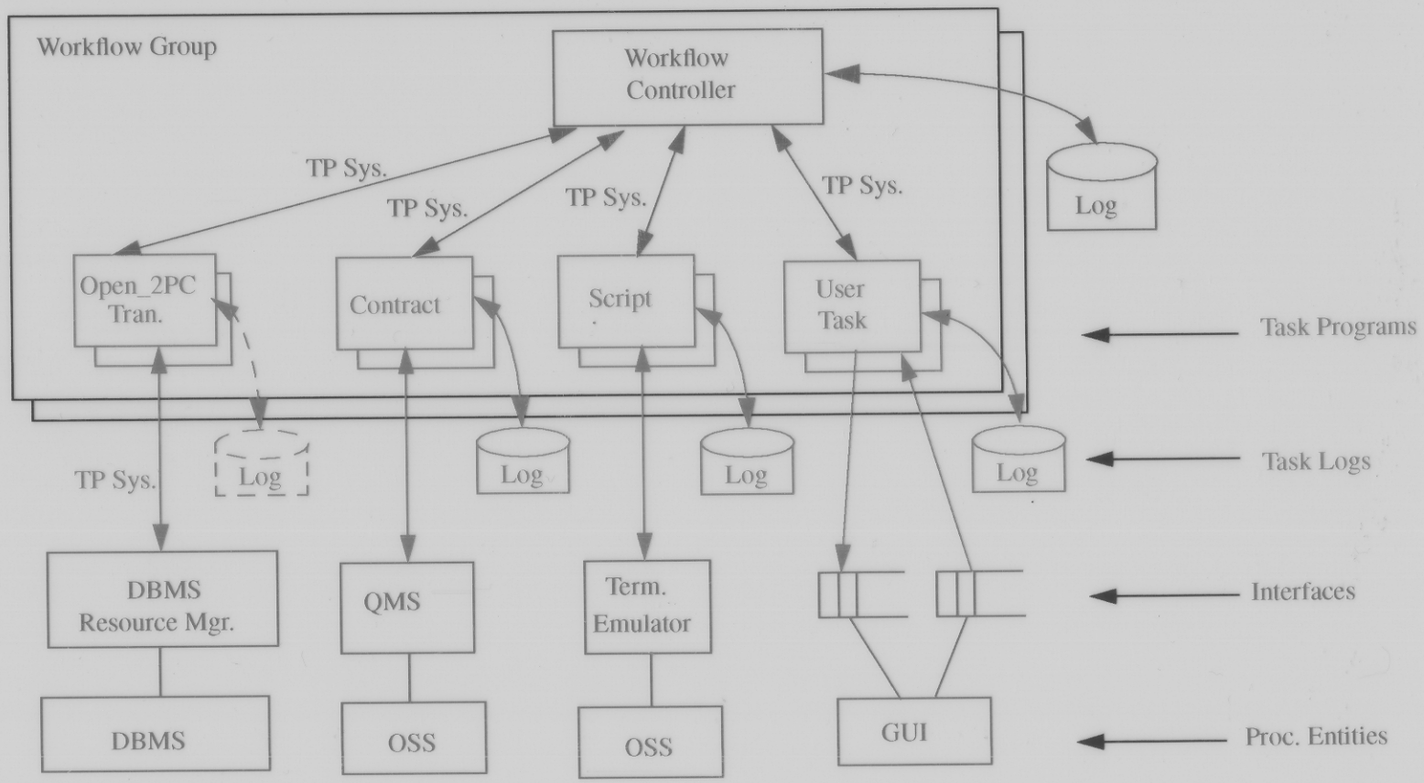
- Task types: task structures, data input/output
- Task classes, Task instances
- Inter-task dependencies (logical error handling)
- Data exchange statements
- Filter (interface def.)

Component of TSL (partial)

- processing entity specific statements
- statements for revealing task structures
- statements for identifying interfaces and dealing with systems errors

[Sheth and Krishnakumar 94]

Run-time Architecture (METEOR Approach)



[Sheth and Krishnakumar 94]

Scheduling Approaches

- Based on Petri-net Models [Elmagarmid et al 90]
- Executor for Flex. Trans. in a logically parallel language L.0 [Ansari et 92]
- Interpreter of MDB transaction specification Language (VLP) [Keuhn et al 91]
- Interpreter of ECA rules [Dayal et al 92]
- Games vs. Nature [Rusinkiewicz et al 92]
- Fine-state Automata [Jin et al 93]
- Scheduling and enforcing intertask dependencies using temporal propositional logic [Attie et al 93]

[Rusinkeiwicz and Sheth 94]

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 until e_2 is submitted, reject e_1 if task 2 terminated without submitting e_2],

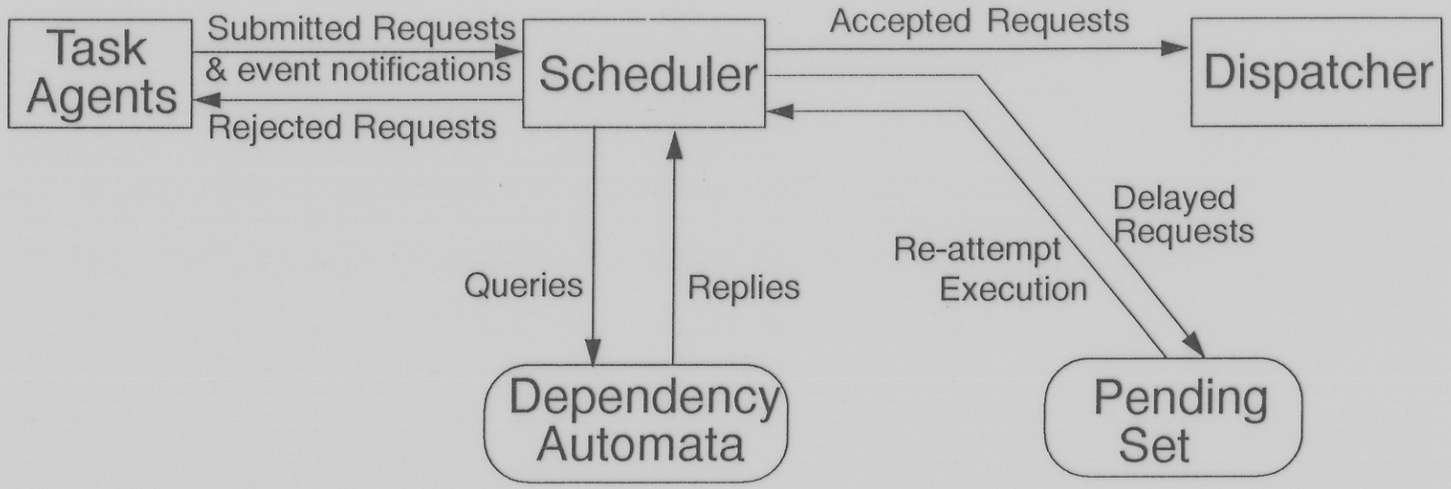
    or forcible(e_2) [force execution of e_2 when e_1 is accepted for execution].
  - $e_1 < e_2$ is run-time enforceable if

    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]

Execution Model
(a centralized approach)



- (Partially) distributed [Jin et al] scheduler per workflow, CC & R
- Ambulatory approach [INCA - Barbara ...]

[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].

[Sheth and Rusinkiewicz 93]

Application/Task and System Semantics

Semantics (Application/Task, System)	Impact (CC: Con. Control, R: Recovery)
limited commutativity (appl)	fewer exclusive locks (CC)
relaxed isolation (appl)	no global commitment (CC)
order preservation + rigorousness (sys)	early release of locks (CC)
idempotency (sys)	resubmit transactions (R)
+ monotonicity	roll-forward recovery (R)

[Jin et al 93, Jin et al 94]

Prototype 1: PROMPT

(A system for Processing Multidatabase Transactions)

Approach

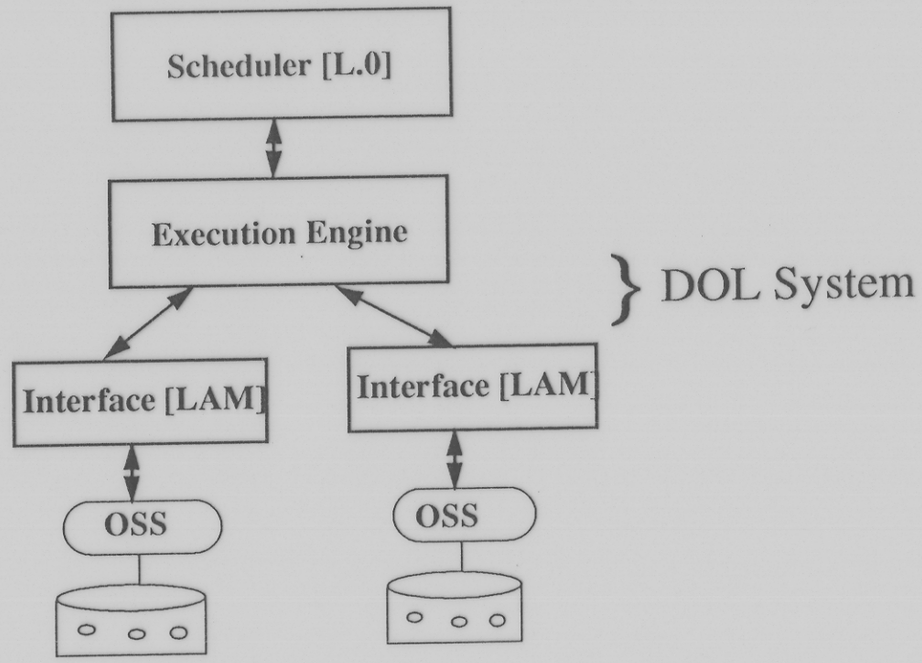
Provide flexible, declarative, high-level flow specification, control, and interfaces using a relaxed multidatabase transactions model and execution environment.

PROMT Components

- Specification: Flexible Transactions for declarative, flexible work-flow specification [Elmagarmid/Rusinkeiwicz et al. 90]
- Scheduling: parallel logic language for controlling execution of concurrent activities (L.0) [Cameron et al. 91]
- Execution/Interfaces: manage execution of transaction on heterogeneous, autonomous systems (DOL System, Narada) [Halabi et al 92]

Prototype 1: PROMT

(A system for Processing Multidatabase Transactions)



OSSs (application systems) emulated using DBMSs.

[Bellcore, UofH: Ansari, et al 92]

Prototype 2 Architecture

User
- initiate new workflow instances
- monitor progress

Application Developer and System Analyst

Workflow initiation, (task input,) Workflow monitoring interface

Workflow Control System

Task Agents

Scheduler

Execution & Event Monitor

Interface (ESS)

Interface (ESS)

Interface (ESS)

Work flow Specification and Simulation Interface

- concurrent work flows
- fault tolerance, recovery

OSS

OSS

Network

Legacy and New Systems

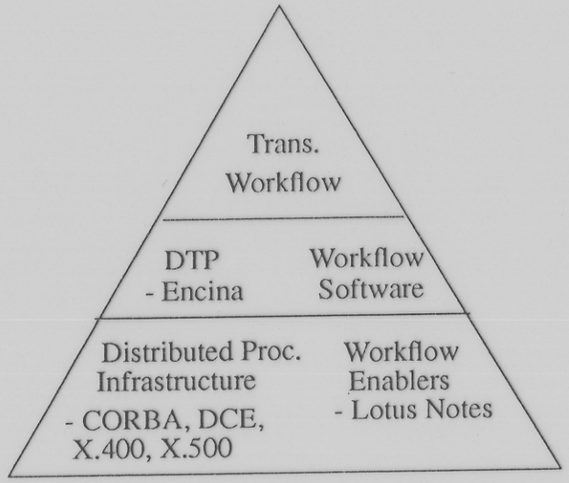
OSS: Operation Support System

[MCC, Bellcore, UofH, ...]

Work in Progress...

- Graphical instantiating and monitoring of workflows [MCC, Bellcore]
- Multidatabase/Semantic Transaction, ETM, ATM *vs.* Transactional Workflow [Breitbart et al 93] [Georgakopoulos, GTE]
- Concurrency Control and Recovery that exploit application and system semantics [Jin et al 93a,b]
- Partially distributed scheduler [one scheduler per workflow] (Jin et al 93b)
- Distributed Scheduler [Singh, MCC]
- Graphical specification and testing of workflows[Georgia]
- Workflow simulation environment [Georgia]
- ...

Technology Perspective
for future commercial directions towards TWF



Comments on Status and Research Challenges

- Model: when to use transactional features, correctness issues
- Language: ease of specification vs features
- Run-time System: error handling, recovery, correctness, performance, heterogeneous objects
- Commercial Products: cross-market strategy
- Standards: plenty of relevant efforts, what is useful now?