TEXAS TECH UNIVERSITY™

Monitoring Data Dependencies to Support Recovery in Concurrent Process Execution*

Susan D. Urban
Department of Computer Science

February 6, 2009

*This research is partially supported by NSF Grant No. CCF-0820152.

The Challenge of Concurrent Execution in a Service-Oriented Environment

Serializability

- The concurrent execution of two or more transactions must be equivalent to the serial execution of those transactions
- Two-phase locking and two-phase commit support serializability in controlled distributed environments

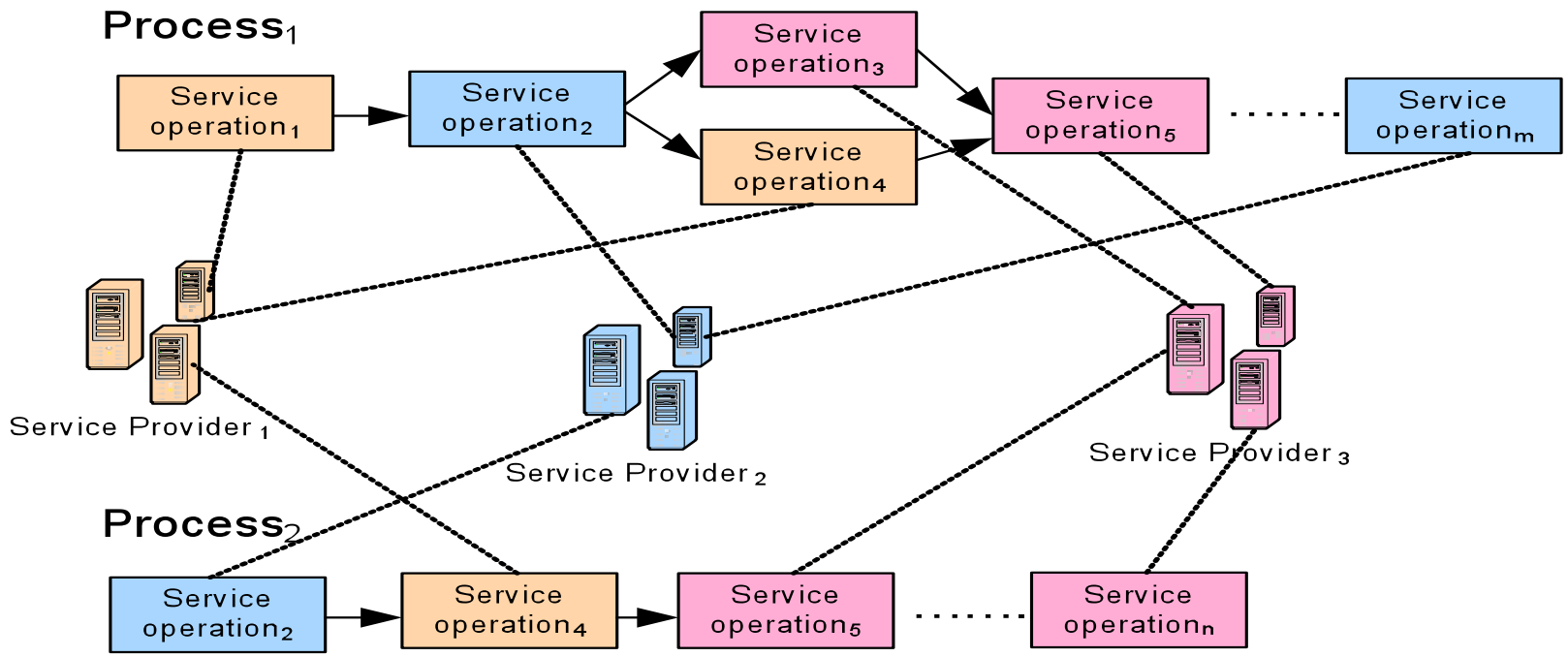
Isolation

- Data changes should not be released before the commit of a transaction
- Lack of isolation leads to cascaded rollbacks when transaction failure occurs.
  - *Transaction A fails and performs rollback*
  - *If transaction B reads modified data from transaction A, transaction B must also rollback*

The problem: Serializability and isolation are not generally applicable to long-running workflow or process scenarios composed of distributed, autonomous services.

- Compensation can be used to logically undo a process
- Compensation does not account for the affect of the failure and recovery process on concurrently executing processes

Concurrent Process Execution Scenario



Scenario

- $Process_1$ fails at service $operation_5$
- Compensation can be executed to restore $Process_1$
- $Process_2$ may be operating with incorrect data

Research Challenges

Harnessing Moore's Law, by Mark Hill

"Our Success in hiding computers when they work brings with it a responsibility to hide them when they fail. Imagine Web Services as available as telephones
….we will have to design systems assuming that they will fail….
we should seek to ensure
that all systems mask almost all of those failures from users."

From *Computer Science: Reflections on the Field*, *Reflections from the Field*, National Research Council of the National Academies, 2004.

Overview of Presentation

Related Work

The DeltaGrid Approach

- Overview of the Approach
- Delta-Enabled Grid Services (DEGS)
- Process Dependency Model
- Service Composition and Recovery Model
- Process Interference Rules and Recovery Algorithm

Implementation, Simulation, and Performance Evaluation

DeltaGrid Research Contributions

Current Directions (NSF Grant No. CCF-0820152)

- The D3 Project: Decentralized Data Dependency Analysis and Recovery for Concurrent Processes

THE REACTIVE BEHAVIOR AND DATA MANAGEMENT RESEARCH TEAM

Past Members from Arizona State University

- Luther Blake (M.S.) *The Design and Implementation of Delta-Enabled Grid Services*, 2006.
- Yang Xiao (Ph.D.) *Using Deltas to Analyze Data Dependencies and Semantic Correctness in the Recovery of Concurrent Processes*, 2006.
- Vidya Gopalan (M.S.) *Simulation and Evaluation of an Object-Oriented Condition Evaluator for Process Interference Rules*, 2008.

Current Team from Texas Tech University

- Ziao Liu, M.S. Student, *Decentralized Data Dependency Analysis for Concurrent Process Execution – in progress*
- Le Gao, Ph.D. Student – in progress
- Andrew Courter, B.S./M.S. Student - in progress

http://reactive.cs.ttu.edu

Related Work: Transactions and Workflows

Transactional Workflow

- The ConTract Model (compensation, pre-/post-condition) (Wachter and Reuter 1992)
- METEOR (pre-defined hierarchical error model) (Worah 1997)
- CREW (explicitly specify data dependency) (Kamath and Ramamritham 1998)
- WAMO (automatic exception handling for workflow execution) (Eder and Liebhart 1995)

Exception handling in service composition environment

- Transaction protocols: WS-Transaction (Cabrera et al. 2002)
- Transactional Attitude (Mikalsen, Tai, and Rouvellou 2002)
- Web Service Composition Action (contingency) (Tartanoglu et al. 2003) (Tartanoglu et al. 2003)
- BPEL4WS (Andrews et al. 2003)
- BPML (Arkin 2002)

Our Research

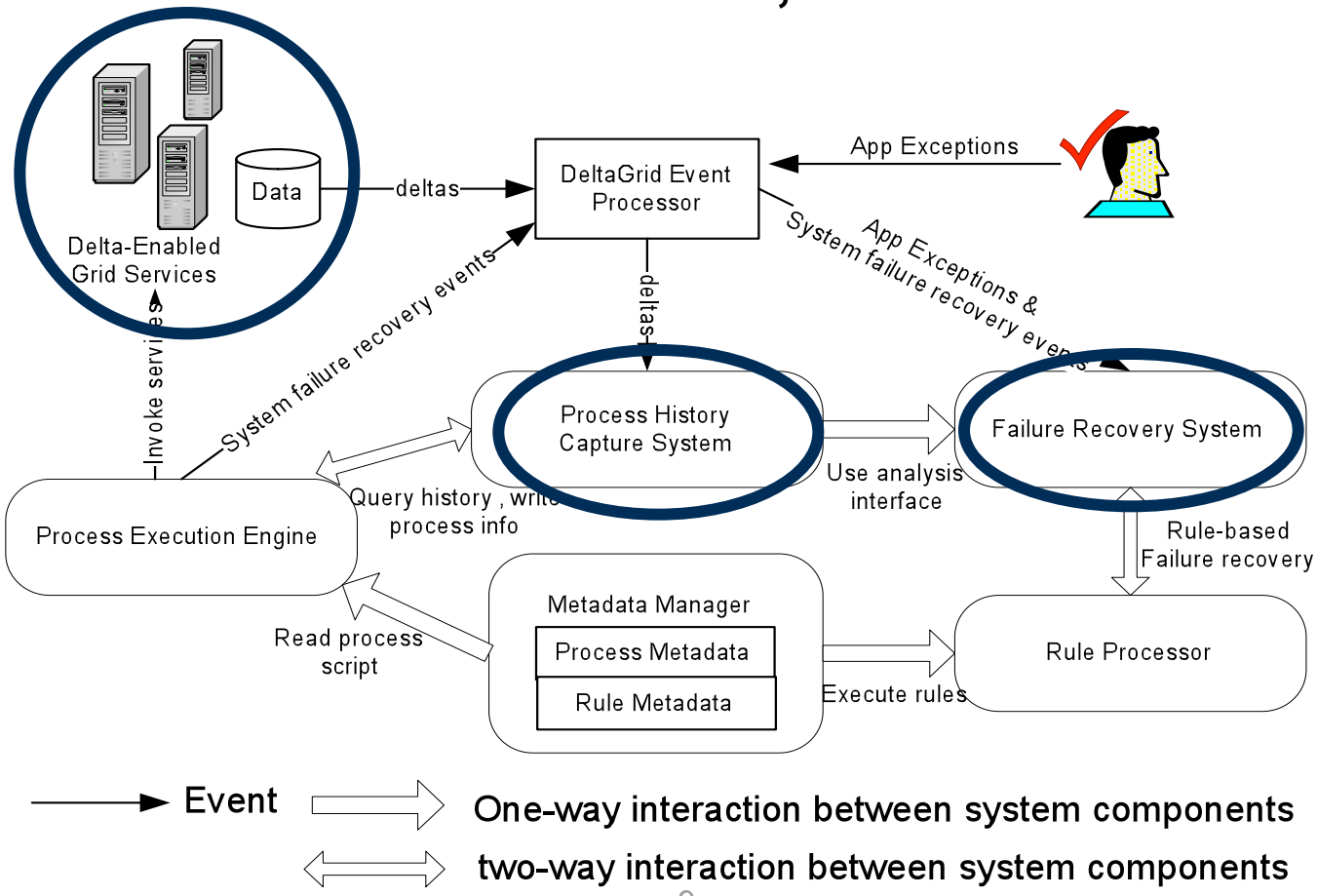
- Supports relaxed isolation and user-defined semantic correctness
- Rule-based approach to resolving failure and recovery impact on concurrent processes.
- Dynamically analyzes data dependencies from streaming database log files.

The DeltaGrid Approach

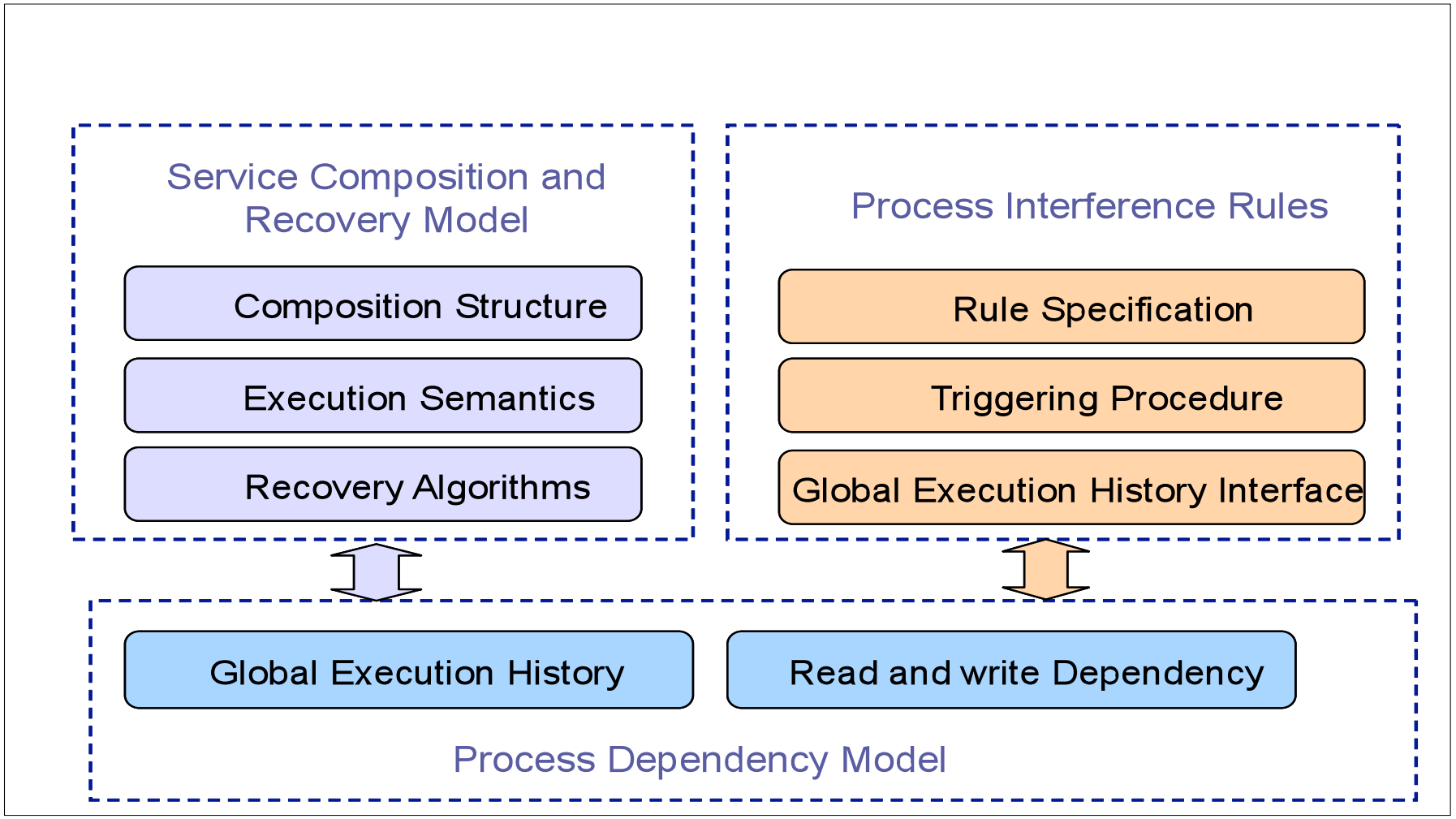
Overview of the Approach

The DeltaGrid Approach

A semantically-robust execution environment for processes that execute over distributed, autonomous services



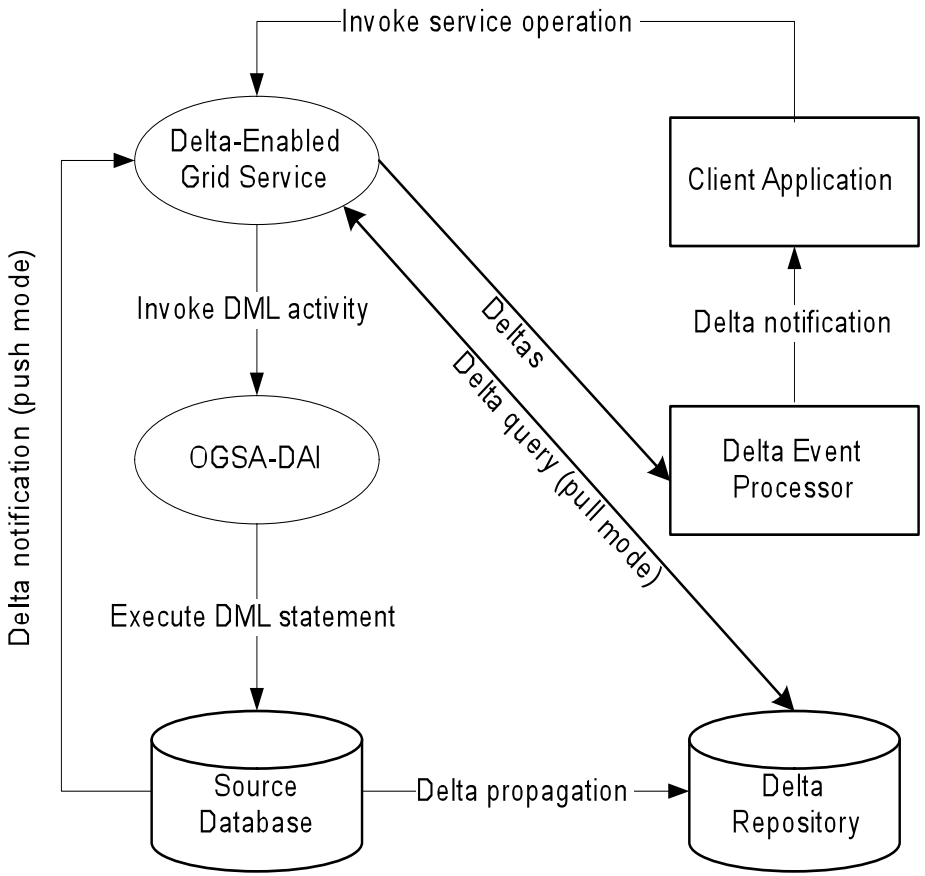
DeltaGrid Abstract Execution Model



The DeltaGrid Approach

Delta-Enabled Grid Services

Delta-Enabled Grid Services



- ***Delta*** – An incremental change in a data element
- Captures data changes using either
  - Triggers
  - Oracle Streams
- Sends deltas back to the delta event processor in either a push or pull mode using XML
- Provides a way to externalize the DB log file as a stream of data change events

Triggers vs. Streams

Triggers

- Tightly coupled to update transaction
- Doubles time for update
- Push of deltas is not automatic
- Easy to use but inflexible

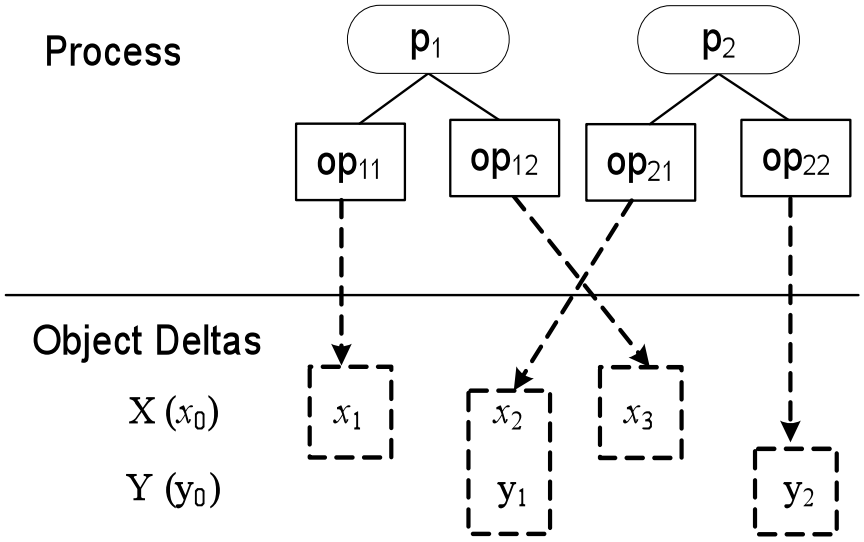
S. Urban, Y. Xiao, L. Blake, and S. Dietrich, Monitoring Data Dependencies in Concurrent Process Execution Through Delta-Enabled Grid Services, to appear *in International Journal Of Web and Grid Services*, 2009.

Oracle Streams

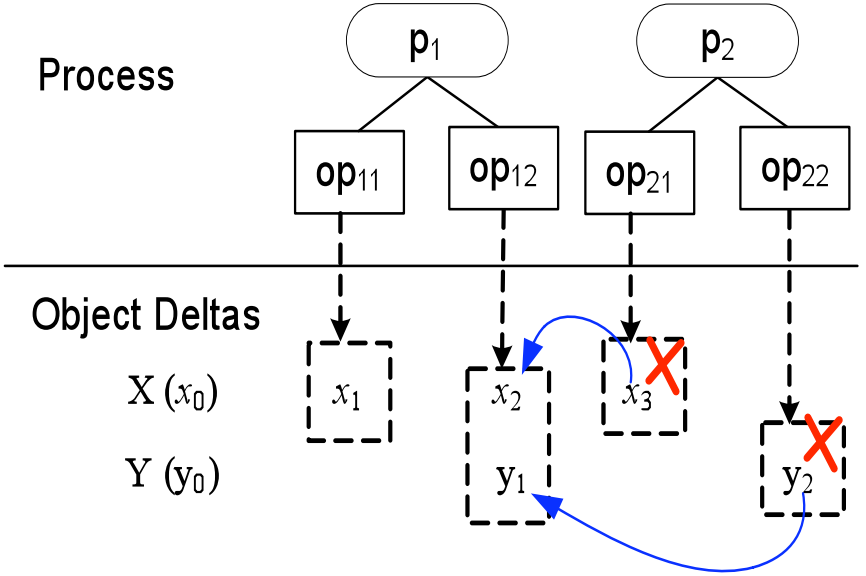
- Decoupled from update transaction
- Offload delta repository to limit affect on updates
- Automatic streaming to multiple destinations
- Complex but versatile

Expanding Investigation to DB2 and SQL Server

Use of Object Deltas



Dynamically analyze *data dependencies* in concurrent process execution to identify *process interference* when failures occur.



Delta-Enabled Rollback (DE-rollback) can be used if recoverability conditions are satisfied.

The DeltaGrid Approach

Process Dependency Model

Write/Potential Read Dependency

Write Dependency

- Process-level

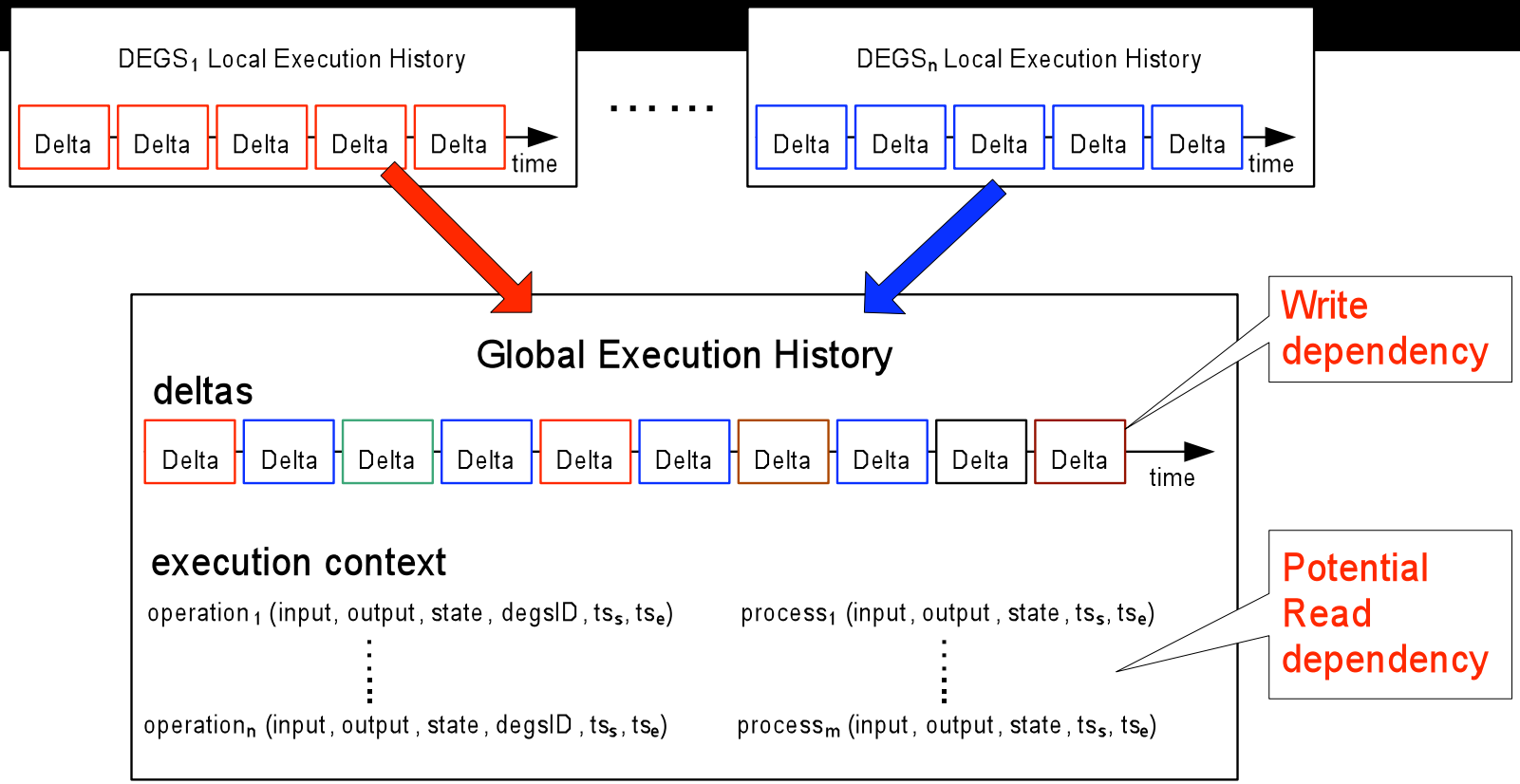
  A write dependency exists if a process p_i, writes a data item *x* that has been written by another process p_j before p_j completes ($i \neq j$).
- Operation-level
- Write dependency set

Potential Read Dependency

- Process-level

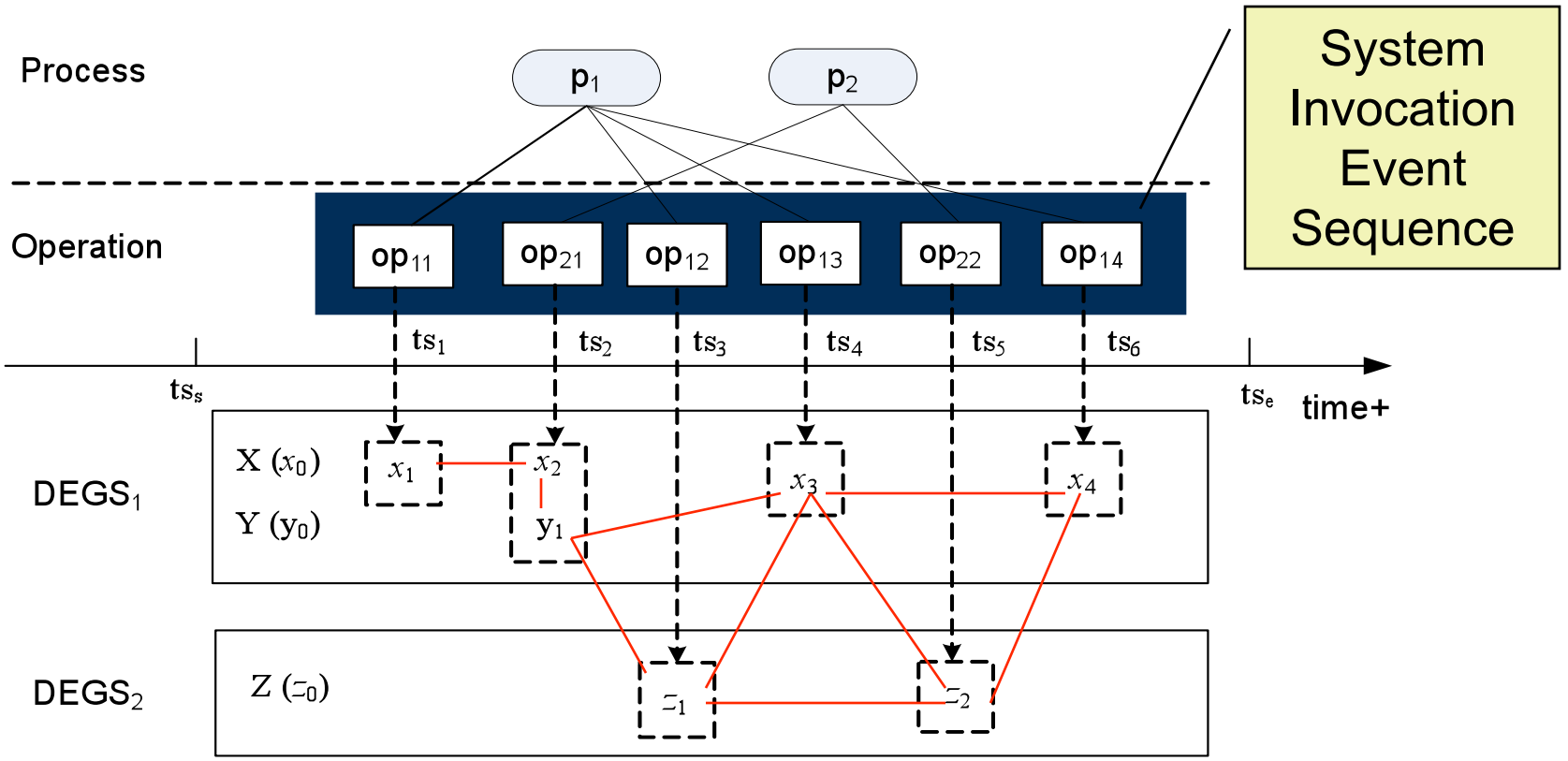
  A read dependency exists if a process p_i, read a data item *x* that has been written by another process p_j before p_j completes ($i \neq j$).
- Operation-level
- Potential read dependency set

Global Execution History



Y. Xiao and S. Urban, Process Dependencies and Process Interference Rules for Analyzing the Impact of Failure in a Service Composition Environment, *Journal of Information Science and Technology*, 2008. Special issue from *10th International Conference on Business Information Systems*, Poznan, Poland, 2007.

Process Execution Scenario



Local Execution History of DEGS1

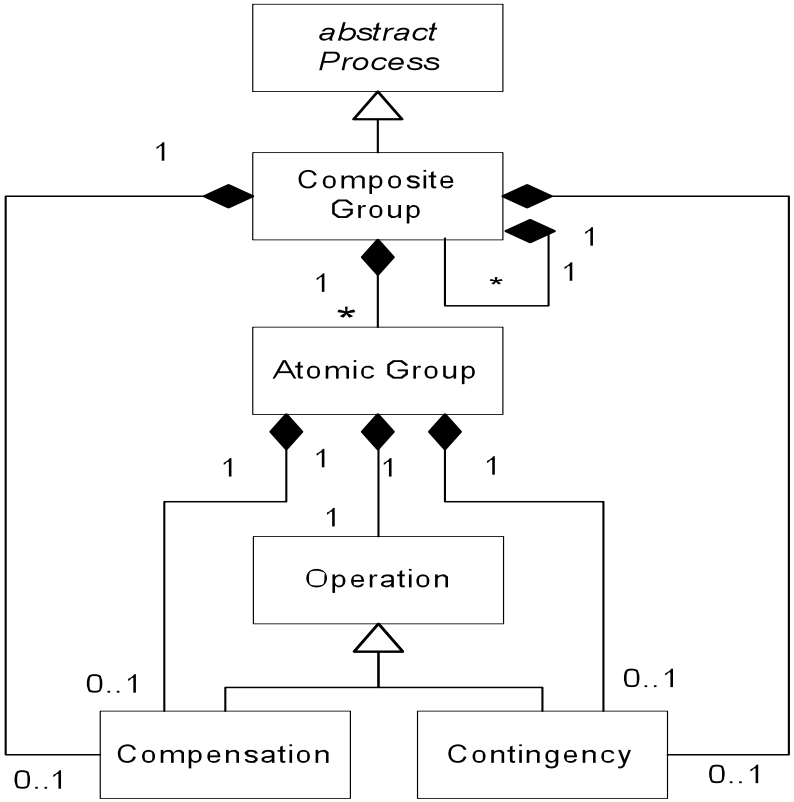
Global Execution History

Local Execution History of DEGS2

The DeltaGrid Approach

Service Composition and Recovery Model

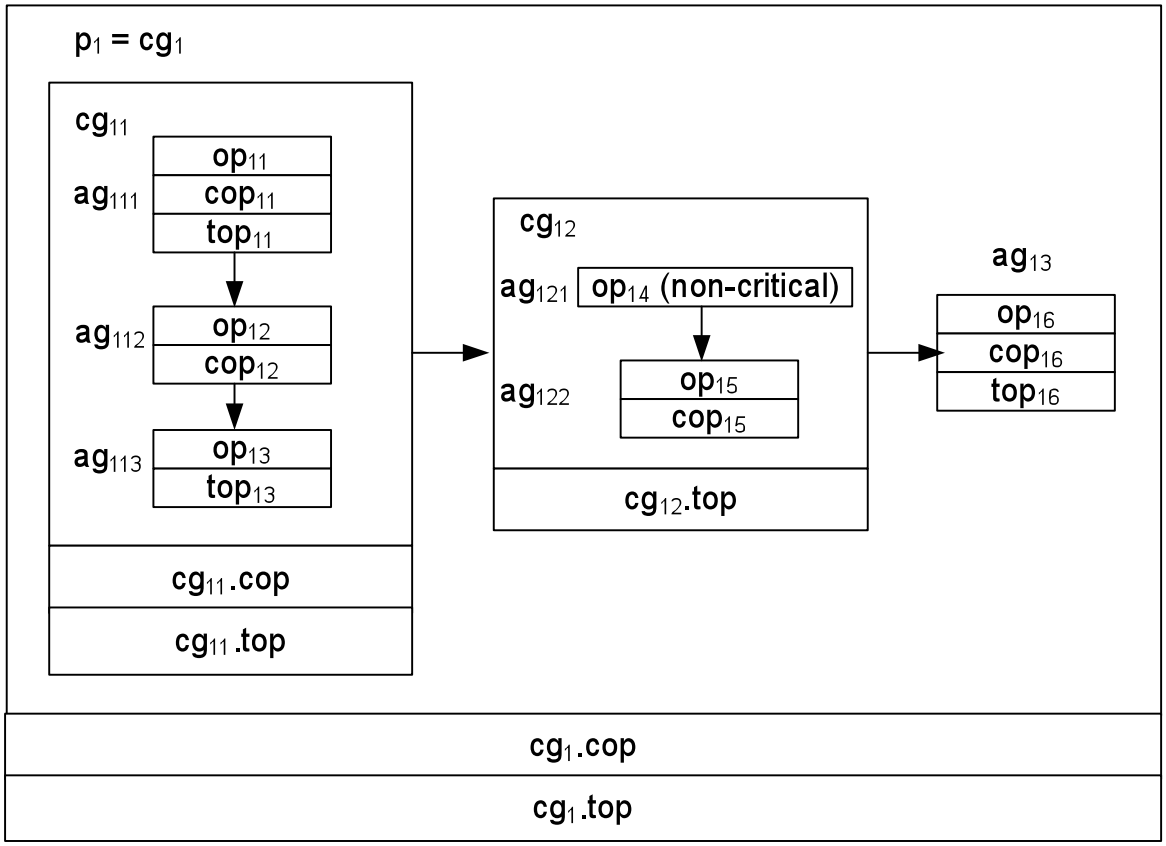
Service Composition Structure



Execution Entities:

- Operation
- Compensation
- Contingency
- Atomic Group
- Composite Group
- Process

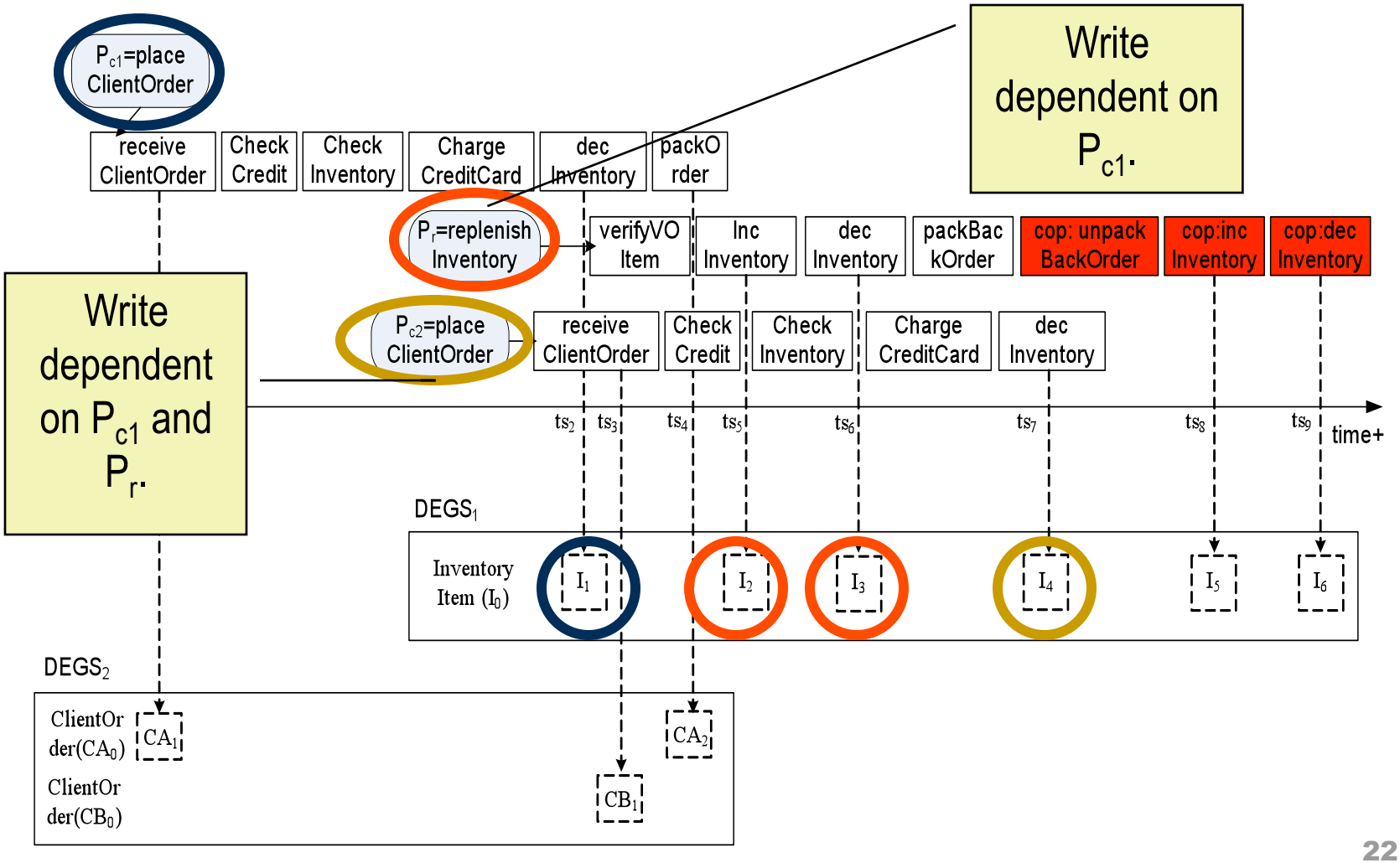
Abstract Process Definition Example



Yang Xiao and Susan D. Urban, The DeltaGrid Service Composition and Recovery Model, to appear *International Journal of Web Services Research*, 2009.

- Atomic Group
  - Compensation
  - Contingency
- Composite Group
  - Deep/Shallow compensation
  - Contingency
- Supports DE-Rollback
- Provides state diagrams and algorithms for recovery semantics of the service composition model (single and concurrent execution cases)

Example: Process Interference Caused by Write Dependency



The DeltaGrid Approach

Process Interference Rules and Recovery Algorithm

PIR Specification

create rule	*ruleName*
event	*failureRecoveryEvent*
define	[*viewName* **as** <OQL expression>]
condition	[**when** condition]
action	recovery commands

event: <processName>ReadDependency(p_f, rdp)
<processName>WriteDependency(p_f, wdp)
define: query over the global execution history interface
condition: determine if process interference exists
action: deepCompensate/re-execute process
post-commitRecover/re-execute operation

Process Interference Rule Example

Compensation of replenishInventory removed inventory items needed in placeClientOrder

```
Create rule     inventoryDecrease

Event           placeClientOrder WriteDependency (failedProcess, wdProcess)

Define          decreasedItems as
                select   fd.old
                from     fd in failedProcess.getDeltasByRecovery ("InventoryItem", "quantity")
                group by fd.old
                having   sum(fd.newValue – fd.oldValue) < 0

Condition       when     exists decItem in decreasedItems:
                         decItem in
                         (select  d
                          from    d in wdProcess.getDeltas("InventoryItem", "quantity

Action          deepCompensate (wdProcess);
```

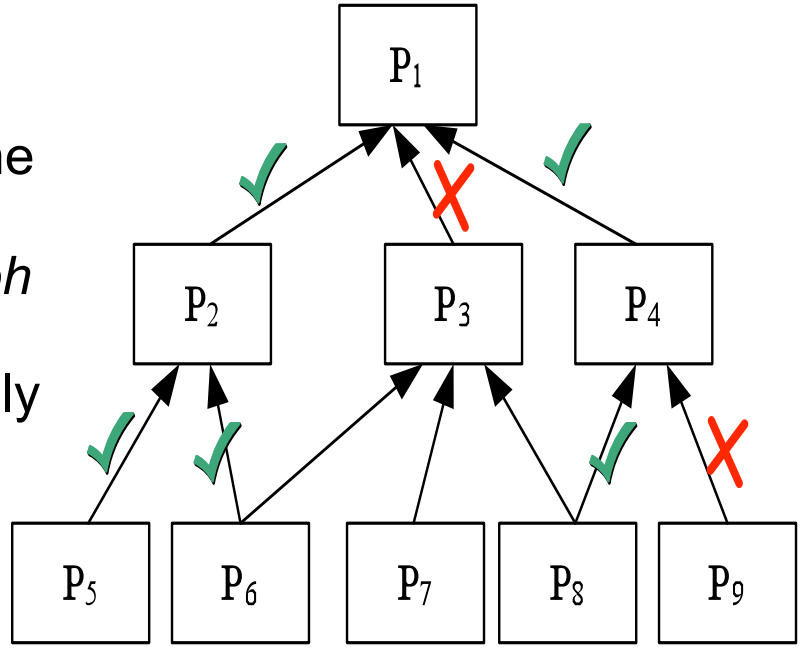
Triggered after failure recovery of failedProcess

Query deltas using object model

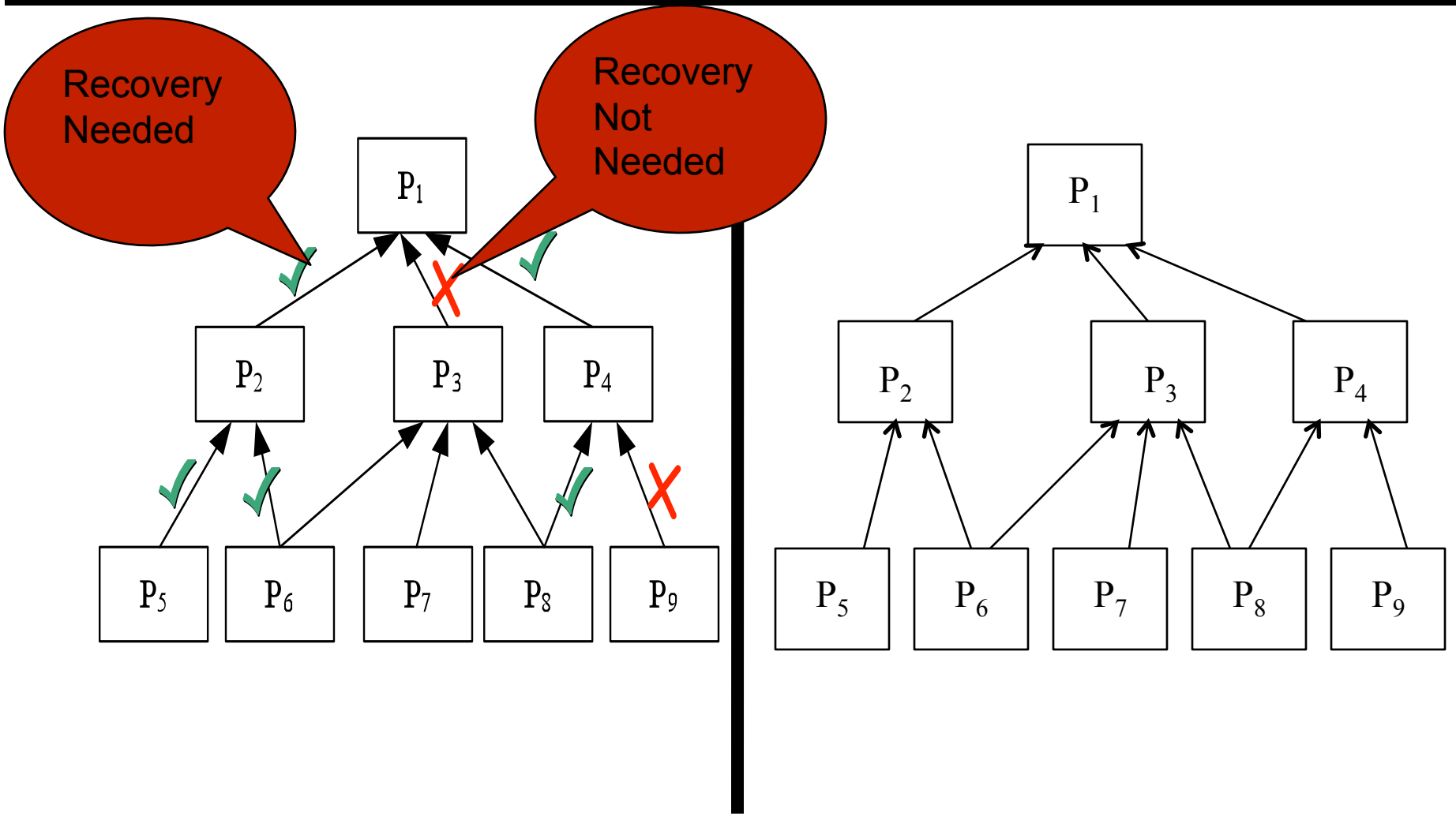
Use application semantics to determine if process interference exists

Concurrent Process Recovery

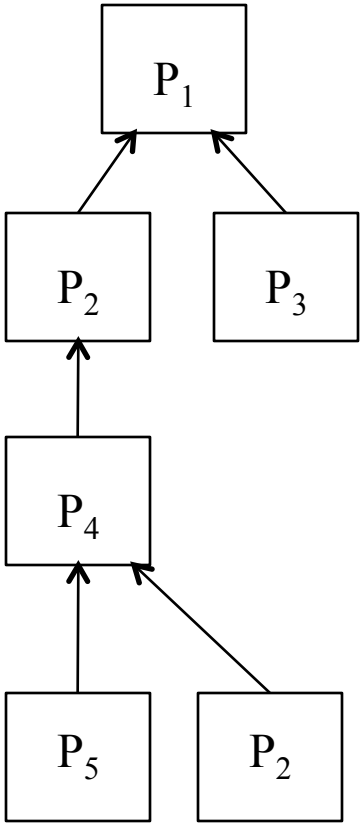
- Execution queue holding active processes
- Generate recovery commands for the failed process p_1
- Generate *process dependency graph* (PDG) for p_1
- Dependent processes are temporarily suspended to evaluate PIRs.
- Breadth-first traversal for PDG and PIR evaluation
  - A process depends on multiple processes
  - A process with PIR evaluated to be false
- Results show the correctness of the PDG formation, the traversal process, use of DE-rollback, and the PIR evaluation process



Cascaded Process Recovery Example



Special Cases to Consider

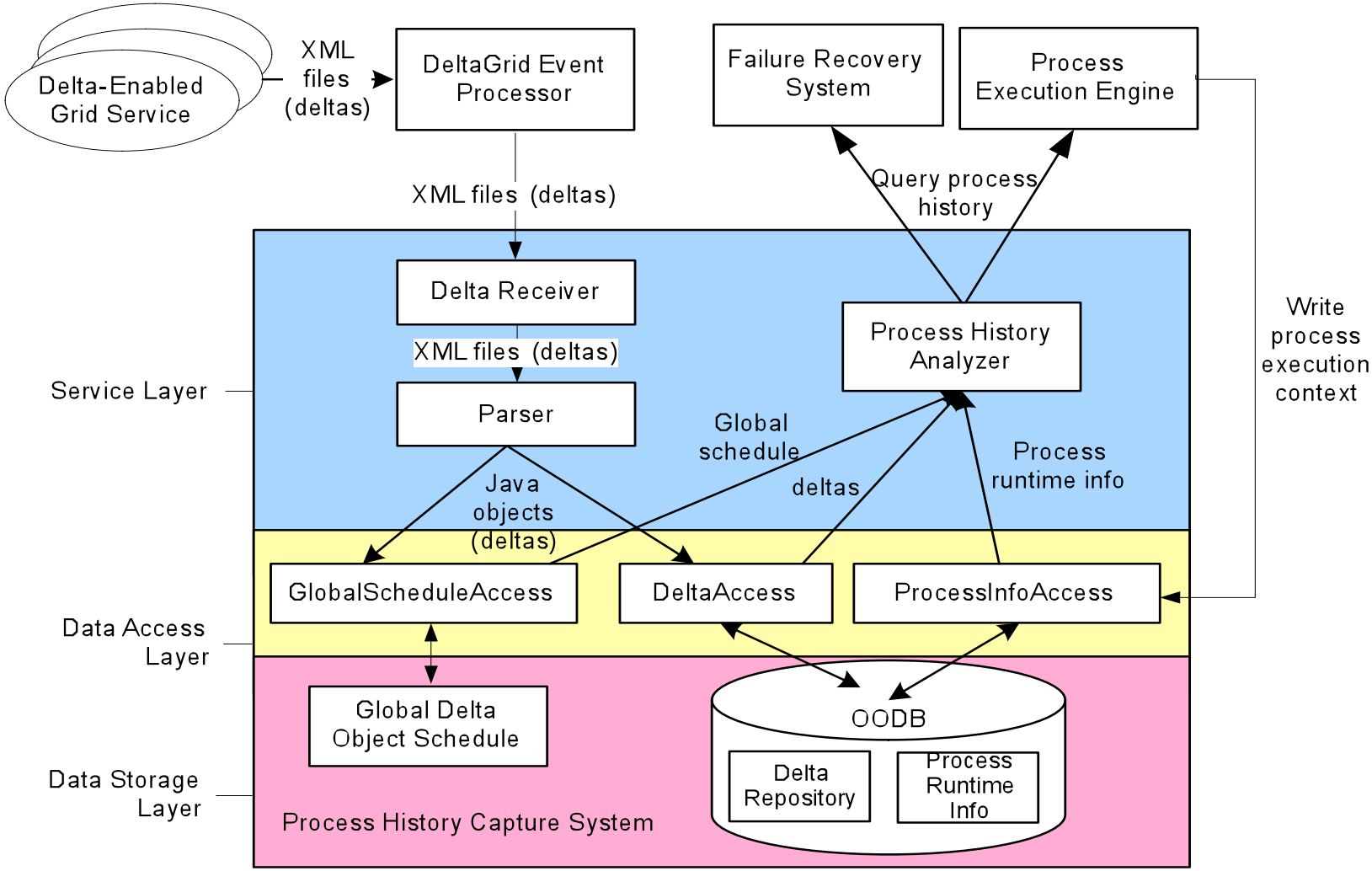


- Handles cyclic dependencies
- Guarantees that updates are not lost in the recovery process.
  - Compensation has higher priority than DE-rollback
  - DE-rollback is only performed if no write dependencies exist.
- Two failed processes p_1 and p_2 can have a common dependent process p_3.
  - Recovery of failed processes p_1 and p_2 are ordered by timestamps
  - If p_3 is recovered with p_1, p_3 does not appear in the dependency graph of p_2 but dependencies introduced by the recovery of p_3 are considered in determining DE-rollback applicability in the recovery of p_2

The DeltaGrid Approach

Implementation, Simulation, and Performance Evaluation

Process History Capture System (PHCS) and Process Recovery System (PRS)



Simulation and Evaluation Framework

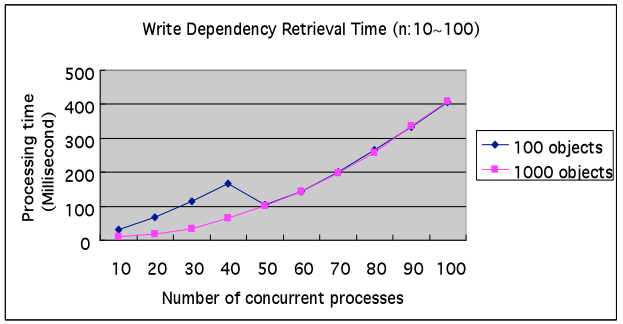
DEVSJAVA (B. Zeigler & H. Sarjoughian)

Implemented PHCS and PRS

Simulated DEGS and Execution Engine

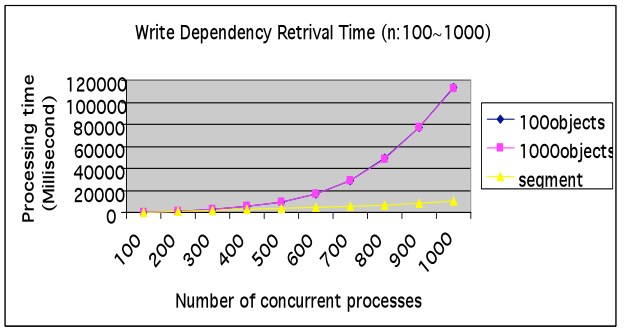
Evaluation Setup for WD Retrieval

- Vary number of concurrent processes (10~100, 100~1000)
- Vary an operation's distribution over objects (100 objects, 1000 objects)



Evaluation Result Analysis

- An operation's distribution over objects does not matter
- Exponential increase without optimization
- Linear increase with optimization based on segmenting the global schedule
- Advocates a distributed PHCS



Other Evaluation Results

Evaluation setup for Recovery Algorithm

- Vary number of concurrent processes (10~100, 100~1000)
- Vary process nesting level (1-5)

Evaluation result and analysis

- Linear increase when the number of concurrent processes grows
  - *Delta parsing/storage time (increases faster than global schedule)*
  - *Global schedule construction time*
  - *Operation-level read dependency retrieval time*
- Exponential increase in PDG construction time with high process density
- Constant cascaded recovery processing time
- Advocates distributed PHCS
  - *Large amount of concurrent deltas*
  - *High process dependency density*

Improved delta object model interface performance through the use of SODA (Simple Object Data Access) interface.

The DeltaGrid Approach

Research Contributions

DeltaGrid Research Contributions

Defined the functionality required for the capture and use of incremental changes to autonomous data sources in a distributed Grid Service environment.

Designed a flexible approach to recovery of service execution failure, providing multi-level protection and maximizing forward recovery

Defined algorithms for analysis of data dependencies among concurrently executing processes based on deltas collected from distributed sites

Designed a rule-based approach for process interference handling based on application semantics

Design, implementation, and evaluation of the DeltaGrid simulation framework

The DeltaGrid Approach

Current Directions: The Decentralized Data Dependency (D3) Analysis Project

The D3 Project

NSF Grant No. CCF 0820152 (Software for Real-World Systems Program)

A Decentralized and Rule-Based Approach to Data Dependency Analysis and Failure Recovery in a Service-Oriented Environment

Objective: To enhance service-oriented environments with theories and methods that support dynamic, flexible, and user-defined approaches to the recovery of failed processes that execute in a loosely-coupled environment without isolation guarantees.

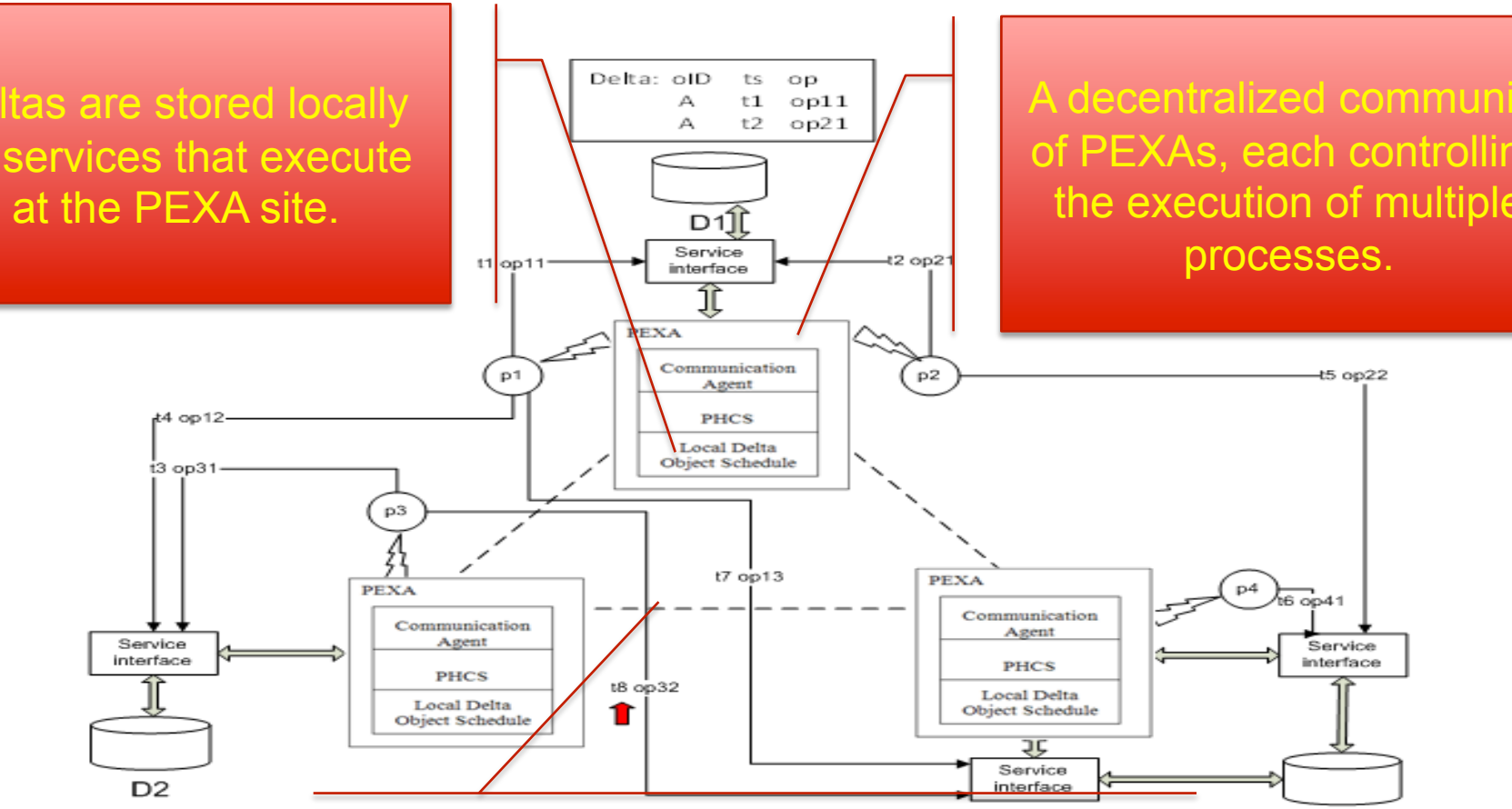
Builds on and integrates three main concepts:

- The DEGS capability of externalizing database log files.
- Decentralized, peer-to-peer techniques for sharing and merging log files.
- Event and rule-driven techniques for dynamic process recovery and exception handling.

Decentralized Process Execution Units

Deltas are stored locally for services that execute at the PEXA site.

A decentralized community of PEXAs, each controlling the execution of multiple processes.



PEXAs communicate in a decentralized manner to dynamically discover data dependencies and to support event and rule driven recovery among concurrent processes.

Research Challenges

Decentralized data dependency analysis

- Representation, communication, correctness, performance

Dynamic aspects of service composition

- Event-driven service composition
- Refinement of process interference rules
- Introduce application exception events and rules

Correctness of execution and recovery with respect to intended user semantics.

- Using formal methods to express execution and recovery correctness in a dynamic, decentralized, concurrent execution environment.

Decentralized algorithms for data dependency analysis, rule execution, and recovery procedures.

Questions?

S. D. Urban, Y. Xiao, L. Blake, and S. Dietrich, Monitoring Data Dependencies in Concurrent Process Execution Through Delta-Enabled Grid Services, to appear *in International Journal Of Web and Grid Services*, 2009.

Y. Xiao and S. D. Urban, The DeltaGrid Service Composition and Recovery Model, to appear *International Journal of Web Services Research*, 2009.

Y. Xiao and S. Urban, Process Dependencies and Process Interference Rules for Analyzing the Impact of Failure in a Service Composition Environment, *Journal of Information Science and Technology*, 2008.

Y. Xiao and S. D. Urban, "Using Data Dependencies to Support the Recovery of Concurrent Processes in a Service Composition Environment," *Proceedings of the Cooperative Information Systems Conference (COOPIS*), Monterrey, Mexico, November, 2008.

Y. Xiao and S. D. Urban. 2007. Process Dependencies and Process Interference Rules for Analyzing the Impact of Failure in a Service Composition Environment, *Proceedings of the 10th International Conference on Business Information Systems*, Poznan, Poland, April 2007, pp. 67-81.

Y. Xiao., S. D. Urban, and N. Liao. 2006. The DeltaGrid Abstract Execution Model: service composition and process interference handling. P*roceedings of the 25th Int. Conference on Conceptual Modeling,* pp. 40-53, Tucson, Arizona.

Y. Xiao, S. D. Urban, and S. W. Dietrich. 2006. A Process History Capture System for Analysis of Data Dependencies in Concurrent Process Execution. P*roceedings of the 2nd Int. Workshop on Data Engineering Issues in E-Commerce and Services,* pp.152-166, San Francisco, California.

H. Ma, S. D. Urban, Y. Xiao, and S. W. Dietrich. 2005. GridPML: A Process Modeling Language and Process History Capture System for Grid Service Composition. P*roceedings of IEEE Int. Conference on e-Business Engineering*, pp.433-440, Beijing, China.

Global Execution History

Delta – An incremental change in a data value.

- $\Delta(oID, a, V_{old}, V_{new}, ts_n, op_{ij})$

DEGS Local Execution History

- $lh(degsID) = <ts_s, ts_e, \delta(degsID)>$
- $\delta(degsID) = [\Delta(oID_A, a, V_{old}, V_{new}, ts_x, op_{ij}) | op_{ij}.degsID=degsID \text{ and } ts_s<=ts_x<=ts_e]$
  ([] indicates a list of elements ordered by timestamp)

Execution Context

- Operation execution context $ec(op_{ij}) = <ts_s, ts_e, Input, Output, State>$
- Process execution context $ec(p_i) = <ts_s, ts_e, Input, Output, State>$
- Global execution context $gec = [ec(entity) | (entity=op_{ij} \text{ or } entity=p_i) \text{ and } (ts_s \leq ec(entity).ts_s < ec(entity).ts_e \leq ts_e)]$

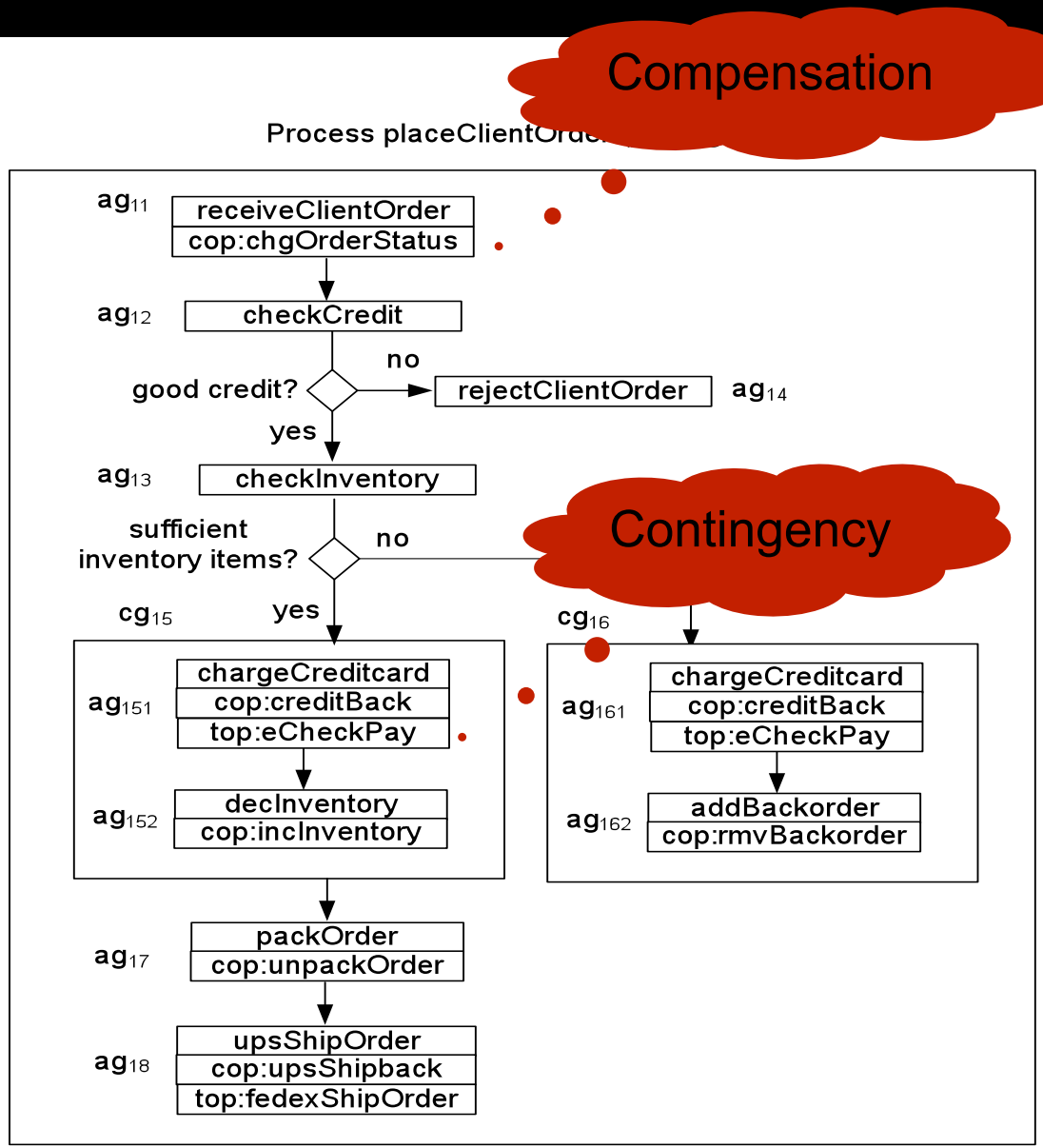
Global execution history

- $gh = <ts_s, ts_e, \delta g, gec>$
- $\Delta g = [\Delta(oID_A, a, V_{old}, V_{new}, ts_x, op_{ij}) | ts_s<=ts_x<=ts_e]$

System Invocation Event Sequence

- $E_{seq} = [e_{entity} | entity = op_{ij} \text{ or } entity = p_i]$

A Process Definition Example



Atomic Group

- Compensation
- Contingency

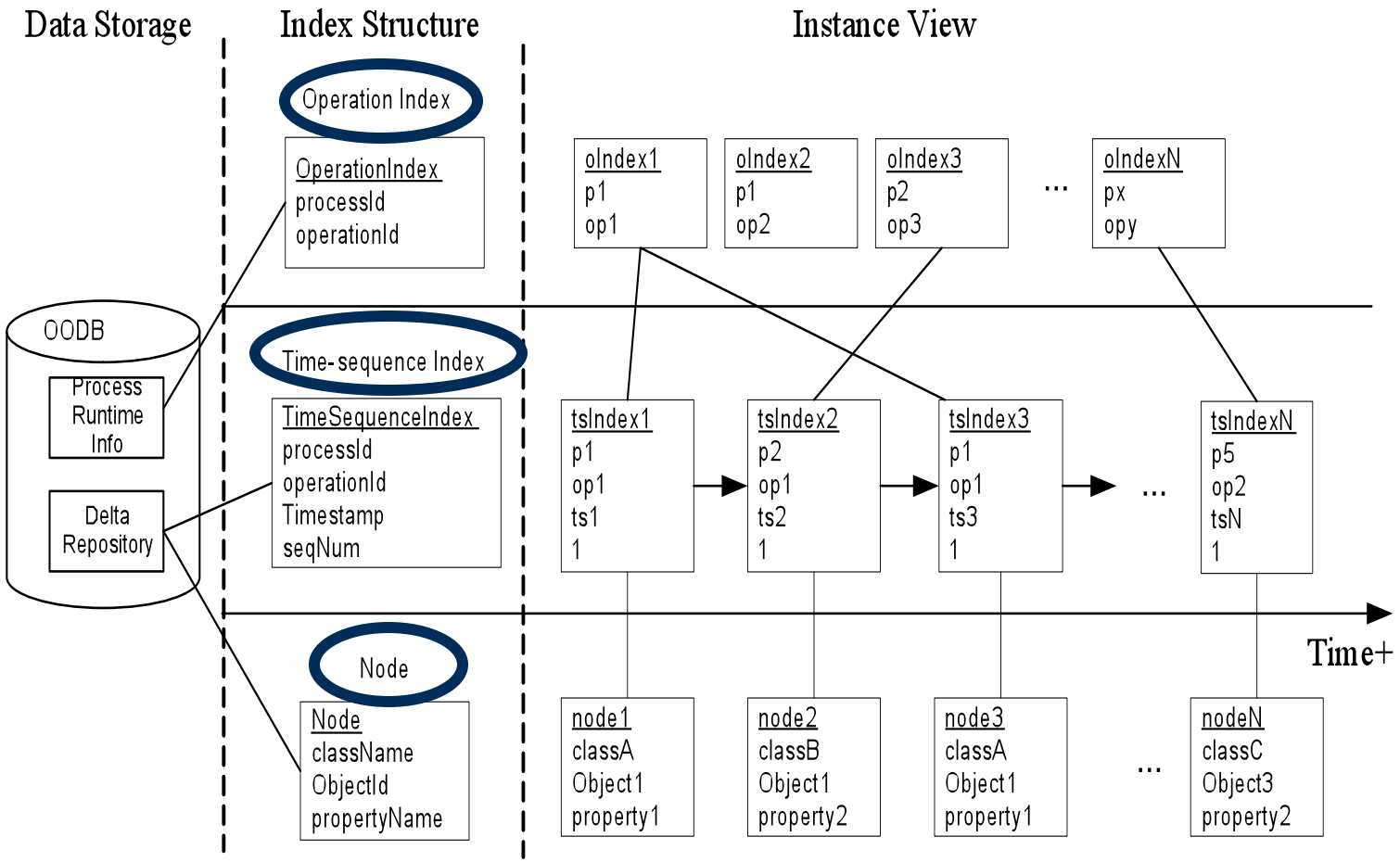
Composite Group

- Deep/Shallow compensation
- Contingency

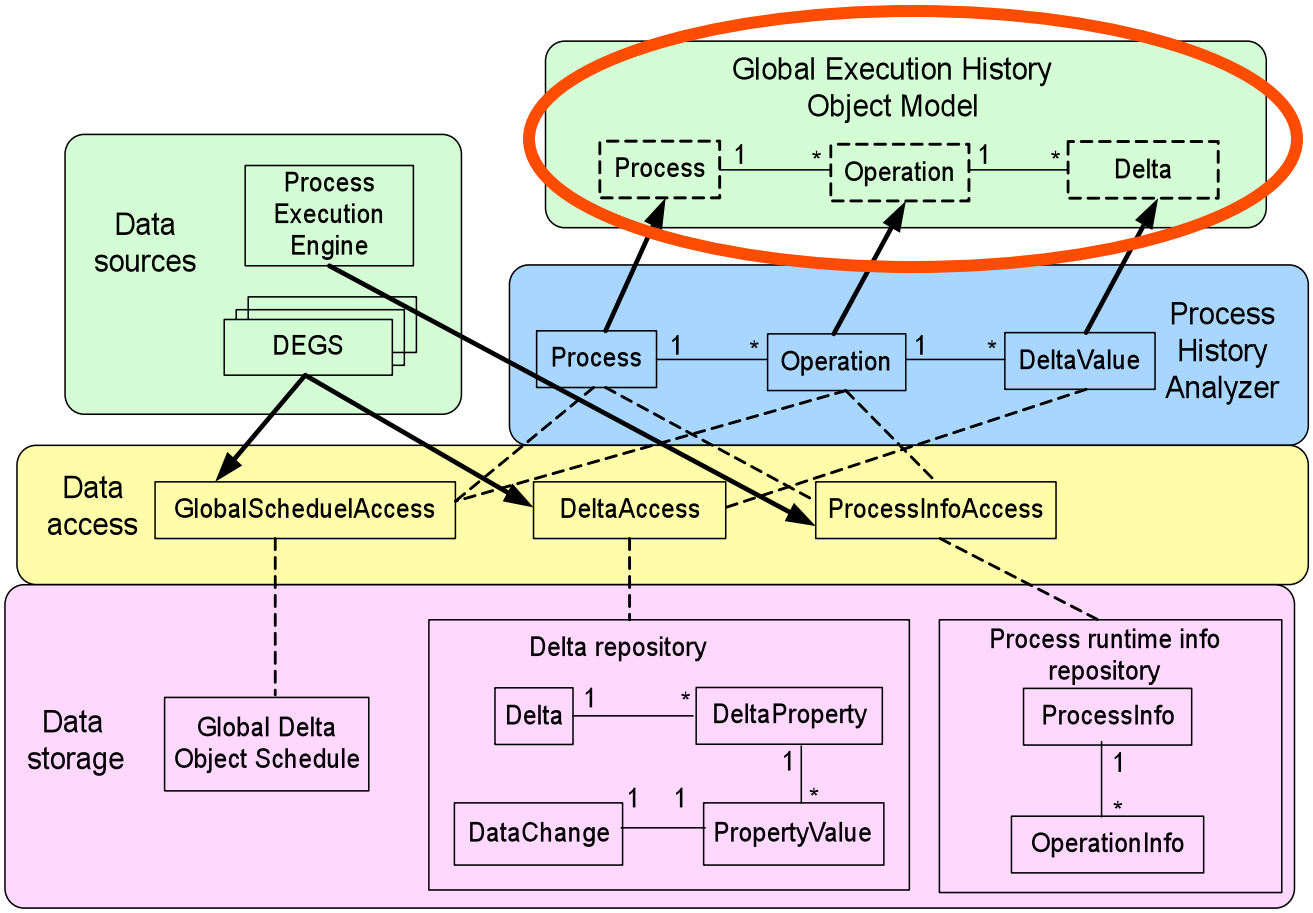
Delta-Enabled Rollback

State diagrams and algorithms for defining recovery semantics of the service composition model (single and concurrent execution cases)

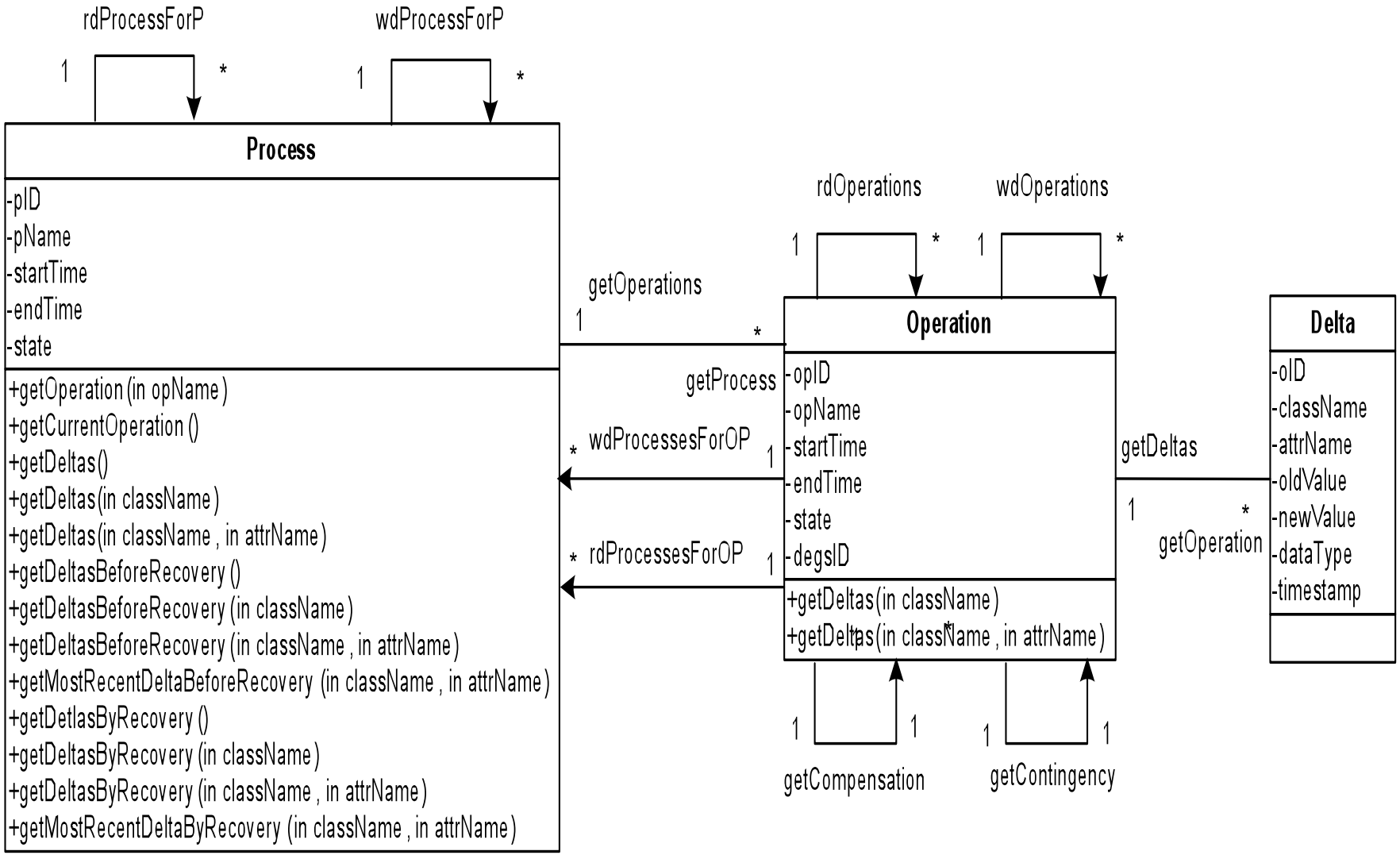
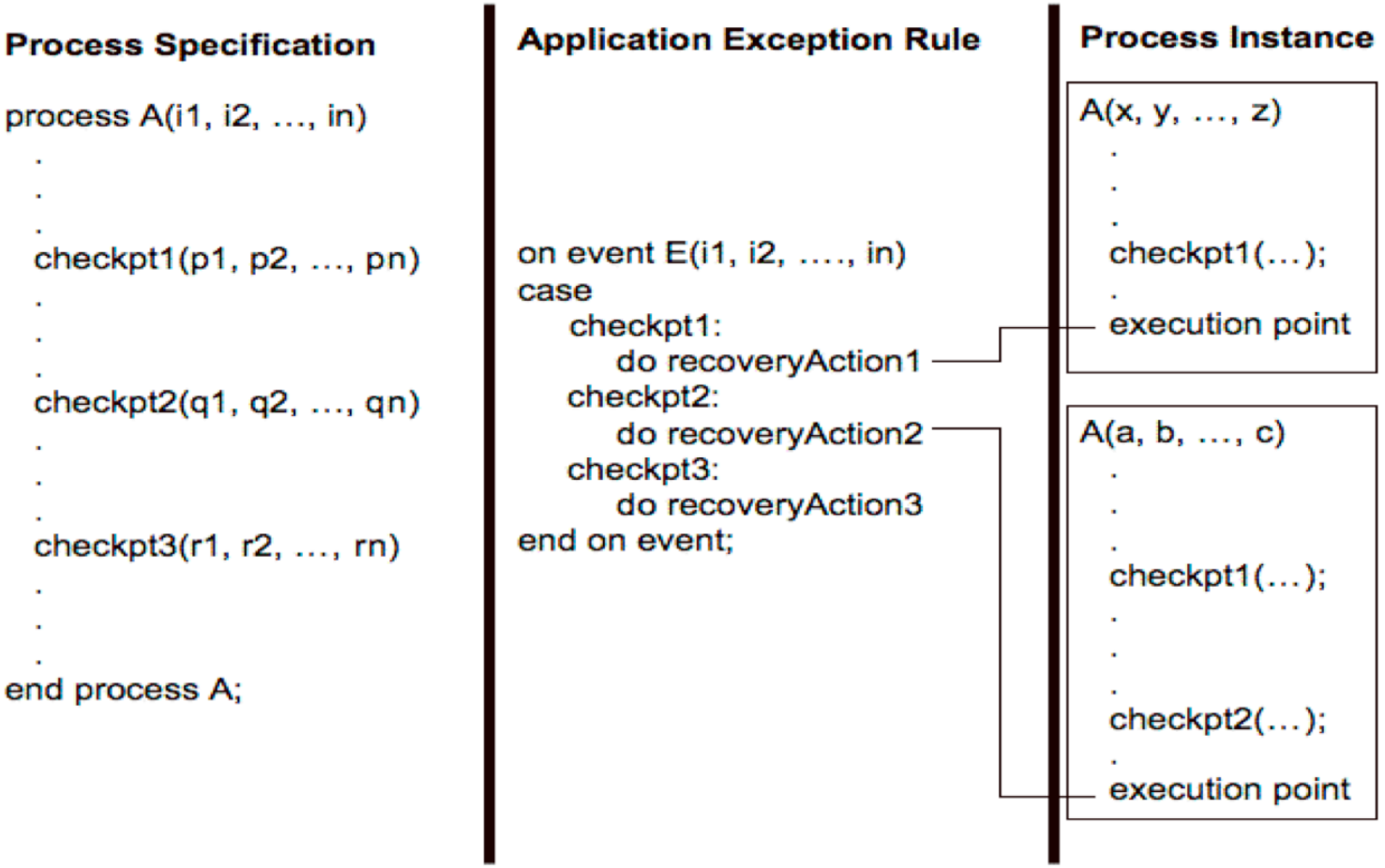
The Global Delta Object Schedule



The Global Execution History Interface Supported by the PHCS



Global Execution History Object Model



Application Exception Rules