Tutorial on Advanced Transaction Models for e-Services

ICWS/SCC/CLOUD/SERVICES 2010

July 5, 2010

K. Vidyasankar, Department of Computer Science, Memorial University, St. John's, Canada

P. Radha Krishna, SET Labs, Infosys Technologies Limited, India

Collaborator: Kamalakar Karlapalem, IIIT-H, Hyderabad, India

Motivation

- Tasks in e-services can be simple or complex.
- Complex tasks are due to:
  - Tasks are inherently complex;
  - Infrastructure support is heterogeneous and loosely coupled;
  - Correctness criteria for successful completion of the task are not clear and can change dynamically;
  - Stringent intra-task dependencies exist.

Motivation - 2

- Participants are autonomous and all participants in a transaction may not have to see the same outcome.
- In B2B relationships, hierarchies with parent-child relationships exist, resulting in nested transactions, and children (sub-transactions) may complete independently of their parents, and hence need to be compensated on the abort of the parent.
- A participant in a service transaction may need to support provisional or tentative state changes (and make them visible to other transactions) during the course of the execution.
- Interoperability issues exist among transactional models.
- There is a large gap between business semantics and application of transactional operations.

Motivation - 3

- Assumptions:
  - Unitary tasks have reasonable transaction properties.
  - Compositions of unitary tasks are in some structured form (sequence, tree, etc.) and their semantics can be understood easily.
  - Support for flexibility in guaranteed execution by means of escalations is available.
  - Task dependencies are usually handled as pre-task or post-task dependencies.
- The question here is how do we provide a software system that can cater to guaranteed executions of such complex services.
- The problem is not trivial !!!.
- The belief is that (database) transactional properties will help.

Database Transactions versus Web Services Transactions

- A database transaction is a partially ordered set of operations.
- A Web service transaction is a partially ordered set of services/activities.
- Differences between operation and activity executions are:
  - Executions of database operations are atomic. Executions of activities are not atomic.
  - Consistency of a (single) database operation is defined very simply. Consistency definition of an activity execution is complex.
  - Each database operation execution has a single successful termination state. An activity execution may have several successful termination states.
  - Roll backs of activity executions are compensation based. Cascade roll backs are common.

Database Transactions versus Web Services Transactions - 2

- Activities in a Web service transaction (composition) are executed typically much more concurrently than database operations. Therefore, complex dependencies occur between executions of different activities.
- Activities are executed in much more autonomous and heterogeneous distributed environments. Concurrency control and recovery mechanisms designed for database transactions need to be modified extensively for Web service executions.
- Multi-level compositions of Web services can be defined analogous to nested transactions. Complexities get compounded in (higher level) composite Web service executions.

Adaptation of Transactional properties

- Historically, transactional properties have been adapted, in some relaxed form, to different execution environments.
- We are now adapting to Services environment.
- This requires identifying and implementing the transactional requirements in Services environment.
- In this tutorial, we describe certain things that have been done and try to give some insight into what should be done.

Agenda

- Brief Introduction to Database Transactions
  - ACID properties
  - Recoverability properties
  - Concurrency control and recovery
- Early Advanced Transaction Models
  - Sagas
  - Nested transactions
  - Relative atomicity
- Execution in Distributed Environments
  - Homogeneous and heterogeneous systems
  - Mobile systems
- Data Item and Operation Granularities
  - Objects and methods
  - Transactional workflows
- Transactional Web Services
  - Multi-level hierarchical composition model
  - Guaranteed terminations
  - Level-wise transactional properties
- Electronic Contracts
  - E-contract basics
  - Multi-level composition model
  - Dependencies
  - Payments
- Recent transaction models for e-services
  - Web services composition with transactional requirements
  - Achieving atomicity using commutativity
  - Protocols for commitment
  - Other proposals

Brief Introduction to Database Transactions

Transaction Definition

- A transaction is an execution of a program.
- A (database) transaction is a partially ordered set of (atomic data) operations.
- The properties associated with a transaction are:
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- These are called ACID properties.

Transaction Properties - 1

- Atomicity
  - Refers to all-or-nothing property.
  - When the execution is complete and successful, the transaction is committed.
  - Otherwise, it is aborted:
    - partial execution, if any, is rolled back. This is called backward recovery.
    - Forward recovery, which refers to completing a partial execution successfully, is also possible some times.
- Consistency
  - Transaction program is correct.
  - Each transaction, when executed alone and to completion, is assumed to be correct, that is, it is assumed to transform a consistent database state to another consistent database state.
  - It follows that a concurrent execution of several transactions is correct when the execution is serializable, that is "equivalent" to some serial execution of the same transactions.

Transaction Properties - 2

- Isolation
  - Refers to the property that the effect of each transaction is the same as when it is executed alone, in spite of possible interleaving of the steps of other transactions.
  - Intermediate states of executions (results) are not available to other transactions.
- Durability
  - This is the guarantee that the effects of a committed transaction persist in the database.
  - This must hold in spite of failures in the system.

Transaction Management – Concurrency Control

- For better utility, transactions are executed concurrently.
- Interleaving of the steps of the transactions is controlled, so as to get a serializable execution.
- Different concurrency control methods have been designed:
  - Two-phase locking,
  - Timestamp methods,
  - Optimistic approach, and
  - several mixed methods.

Transaction Management - Recovery

- Commitment of a transaction is not done atomically.
- Commit response is usually given first and persistence is ensured afterwards, by storing the results in stable database.
- During normal execution:
  - The effects of the committed transactions must be preserved and those of aborted transactions removed. Redo is done for the former, and undo for the latter.
- After system failures:
  - The effects of the committed transactions are restored.
- Logs (log buffers and stable logs) are used typlically.

Recoverability Properties

- To facilitate recovery, execution and/or commitment of transactions may be delayed until some other transactions commit.
- With only read/write operations, for two transactions T and T′:
  - Recoverability implies that if T′ read a value written by T, then T′ cannot commit until T commits.
  - In the above case, if T aborts, then T′ will be aborted too. To avoid such cascaded abort, T′ should not be allowed to read the values written by T until T commits.
  - Strictness means T′ cannot write an object that T also writes until T commits. This simplifies recovery mechanism.
  - Rigorousness implies T′ cannot write an object that T reads until T commits.

Relaxing the ACID properties

- Quite early on, the ACID properties were recognized to be too strict for several applications.
- Relaxations of the properties were proposed.
- Relaxations were to address issues such as:
  - long running transactions;
  - Execution in non-centralized database systems; and
  - Semantics of the transactions.
- Both concurrency control and recovery aspects were considered.
- This affects atomicity, consistency and isolation.

Early Advanced Transaction Models

Sagas

- Proposed by [Garcia-Molina and Salem, 1987].
- A transaction is divided into a sequence of sub-transactions.
- Each sub-transaction is allowed to commit individually:
  - When committed, their effects are visible to all transactions.
- If some sub-transaction has to be aborted, then the whole transaction is aborted:
  - The already committed sub-transactions are rolled back, in reverse order, by executing compensating transactions.
  - Other transactions might have seen the effects of these sub-transactions in the mean time.
- Thus, consistency and isolation requirements are relaxed; atomicity requirement is not.

Nested Transactions

- Proposed first by [Moss, 1981] and refined by others.
- Transactions are decomposed into sub-transactions hierarchically:
  - A (root) transaction is decomposed into sub-transactions, each sub-transaction may be decomposed into further sub-transactions, and so on.
- Different scopes for commit and abort of sub-transactions have been defined:
  - Flat nested
  - Closed nested
  - Open nested
- Global and local isolations come into picture.

Types of Nested Transactions

- Flat nested:
  - Commit of a sub-transaction is local; its effects are visible only to its parent level.
  - Only when the root transaction commits, the effects are visible to other transactions.
  - When any sub-transaction aborts, the entire root transaction is aborted, that is, abort is global.
- Closed nested:
  - Here also, commit of a sub-transaction is local.
  - And, only when the root transaction commits, the effects are visible to other transactions.
  - Here, abort of a sub-transaction is also local; the other sub-transactions and the root transaction are not affected.
- Open nested:
  - Commit of a sub-transaction is global, to the root level.
  - Abort of a sub-transaction is local.
  - When the root transaction aborts, the committed sub-transactions need to be rolled back by executing compensating transactions.

Atomicity Properties of Nested Transactions

- Relaxation of the atomicity property in nested transactions has two distinct characteristics:
  1. An atomic unit may consist of some, not necessarily all, steps of a transaction;
     - For example, a saga is a two-level nested transaction where each bottom level transaction is an atomic unit for every other transaction.
  2. Some steps may constitute an atomic unit to some transactions, not to others
- Characteristic (2) has been generalized in various stages.

Compatible Transactions

- Proposed in [Garcia-Molina, 1983]
- These are a set of transactions whose steps can interleave arbitrarily.
- If T and T′ are not compatible, then the entire transaction T is an atomic unit of T′, and vice versa.
- If T and T″ are compatible, then each step of T is an atomic unit for T″, and vice versa.
  a) the steps of T can interleave with those of T″ arbitrarily and
  b) any number of steps of T″ can be executed after any step of T.
- These properties are constrained in other notions.

Relative Atomicity

- In the relative atomicity notion of [Farrag and Ozsu, 1989]:
  - T″ is allowed to interleave only at certain points in the execution of T, defined as the breakpoints of T with respect to T″; but, whenever T″ is allowed, any number of its steps can be executed.
  - Breakpoints of T with respect to T″ may be different from those of T with respect to another transaction, and hence the term relative atomicity.

Example

- **RESERVE-Transaction:**

```
Step 1 : if Res < Total
            then begin
                Res ← Res + 1
                add new guest $g_i$ to database;
              end
            else exit; (* i.e., terminate transaction *)
Step 2 : Find a (free) room $r_k$ with ST($r_k$) = A;
Step 3 : RM($g_i$) ← $r_k$;
         ST($r_k$) ← U;
```

- **CANCEL-Transaction:**

```
Step 1 : if RM($g_i$) = $r_k$
            then begin
                ST($r_k$) ← A;
                Res ← Res – 1;
                remove guest $g_i$ from database;
              end
            else exit; (* i.e., terminate transaction *)
```

Example – ATOMIC UNITS

- RESERVE–Transaction: T_i : $O_{i\,1}$ $O_{i\,2}$ $O_{i\,3}$
  - The atomic units of T_i relative to another RESERVE–Transaction T_k are [$O_{i\,1}$] and [$O_{i\,2}$, $O_{i\,3}$]
    - i.e.- a break point is defined between $O_{i\,1}$ and $O_{i\,2}$
    - the steps of T_k are allowed to occur between $O_{i\,1}$ and $O_{i\,2}$, and not between $O_{i\,2}$ and $O_{i\,3}$

  - The atomic units of T_i relative to a CANCEL- Transaction are [$O_{i\,1}$], [$O_{i\,2}$], [$O_{i\,3}$]
    - i.e.- a CANCEL-Transaction's steps can be interleaved anywhere with the steps of a RESERVE-Transaction

Relative Serializability

- With the relative serializability notion of [Krishnaswamy et al., 1997]:
  - The above interleavings are only with respect to “dependent” steps.
  - Nondependent steps are allowed to interleave anywhere in T.
  - The precedence relation among the steps of the same transaction and conflict relations among the steps of different transactions contribute to the dependency.
- With the generalized relative serializability notion of [Vidyasankar, 1998]:
  - the number of steps of T'' that can be executed at the individual breakpoints of T is restricted.

Split-Join Transactions

- Proposed by Pu, Keiser and Hutchinson in 1988.
- An executing transaction may be split into two or more sub-transactions.
- Executing sub-transactions may also be joined together.
- Serializability of the final “compositions” of the transactions is required.
- Advantages:
  - Adaptive recovery - Committing resources that will not change;
  - Added concurrency - Releasing the committed resources or transferring ownership of the committed resources.

Execution in Distributed Environments

(Homogeneous) Distributed Database Systems

- Logically a single database; physically distributed.
- Operations are executed at different sites.
- Transactions are coordinated either centrally or in a distributed manner.
- Concurrency control methods for centralized systems are extended.
- Site failures and communication failures may occur.
- For atomicity, all participating sites commit or all of them abort. Two phase and three phase commit protocols are designed for this.

Replicated Database Systems

- Data may be fully or partially replicated at various sites.
- One logical (read or write) operation entails several physical operations on different physical copies.
- Performance is improved by reducing the number of physical operations. Examples are:
  - write-all, read-one;
  - majority write, majority read;
  - using quorums; etc.
- Performance is improved also by responding earlier and performing some physical operations later.
  - Lazy replication in contrast to eager replication.
  - Serializability requirement is relaxed to eventual serializability or eventual consistency.

Mobile Database Systems

- A non-mobile main database and several cached (partially replicated) databases in mobile hosts.
- The mobile and main databases are synchronized at times of connectivity
- Transactions are executed at individual mobile units.
- They are validated against stationary unit when connected.
- Connectivity of mobile units with the stationary one is infrequent.
- Lazy propagation and eventual consistency are the main characteristics.

Heterogeneous Distributed Database Systems - 1

- Examples are multi-database systems and federated database systems.
- Distribution is logical – database schemas may be different.
- Individual database systems are designed for independent local use, but agree to participate in global applications.
- A global transaction is composed of sub-transactions executed in several local sites.
- Local sites are autonomous – Design, Execution and Communication autonomies.

Heterogeneous Distributed Database Systems - 2

- Global transaction management superimposes local transaction management
  - Submission of local sub-transactions is controlled by global transaction manager.
  - Local sub-transactions may be committed before their global transaction.
  - If the global transaction aborts, then locally committed sub-transactions are rolled back by executing compensating transactions.
- (sub-)transactions are defined as compensatable, pivot, and retriable.
- Global transaction composition is restricted, for example, to have at most one pivot.

Data Item and Operation Granularities

Data Items, Abstract Data Types and Objects

- Simple data items:
  - Simple atomic operations - read and write; then create and delete.
- Sets, queues, trees, hash structures:
  - Non-atomic insert, delete, and search.
  - These operations treated as transactions composed of atomic operations.
- Objects:
  - Non-atomic methods.
  - The methods treated as transactions.

Concurrent executions of non-atomic (operations and) methods -1

- Semantics of the objects and the methods determine allowable sequential specifications of the objects.
- A concurrent execution of the methods is expected to be equivalent to some allowable sequential specification.
- Certain operations were considered to "essentially complete" the execution of the method, and the other operations were executed lazily.
  - An example is deleting an index in B-tree.
  - Merging of the nodes, if necessary, could be done lazily.

Concurrent executions of non-atomic (operations and) methods -2

- Isolation was relaxed, especially for non-essential operations.
- Rollback was limited to undoing the essential operations.
- Notion of "critical" atomic operations was used and serializability of concurrent executions was argued in terms of the critical operations.
  - The critical operations were very much like pivots in multi-database applications.

(Transactional) Workflows

- Operations/methods are replaced by activities.
  - A workflow instance is an execution of a partially ordered set of activities.
  - Activities need not be database related, need not even be electronic.
  - They could be manual.
- Consistency of an execution of an individual activity is determined by application semantics.
- Concurrency and isolation are not main issues.
- Achieving atomicity (all-or-nothing property) in each individual workflow instance is the main concern.
- Backward and forward recoveries are typically manual.
  - Forward recovery is in terms of exception handling.
  - Different successful (or acceptable) terminations are used.

Transactional Web Services

[Vidyasankar and Gottfried Vossen, 2004]

[VV] References

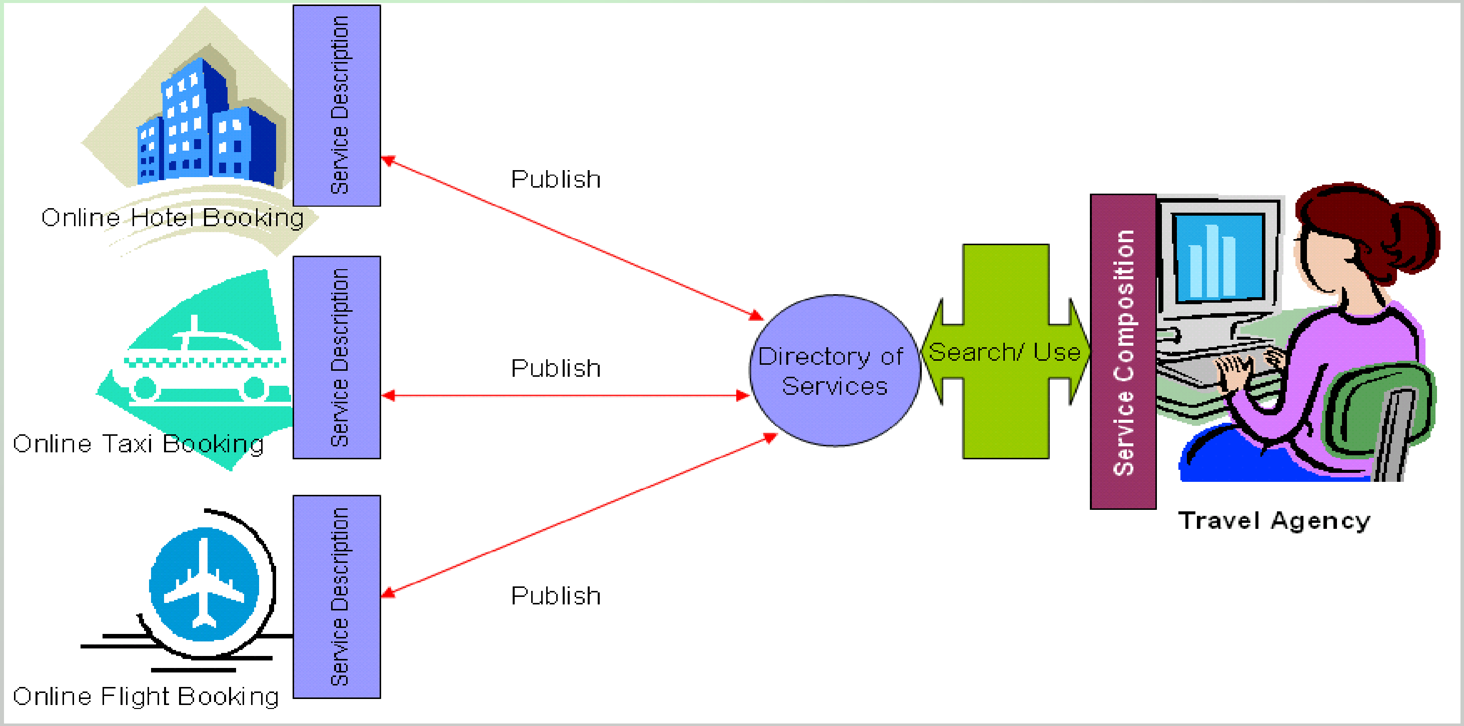
- K. Vidyasankar and Gottfried Vossen [VV]:
  - Multi-Level Modeling of Web Service Compositions with Transactional Properties, Journal of Database Management, Special issue on Service-oriented Computing.
  - A Multi-Level Model for Web Service Composition, Proc. Int. Conf. on Web Services (ICWS 2004), San Diego, July 6-9, 2004, pp. 462-469.

General Vision of Web Services

- Software services can be described in an implementation-independent and semantic fashion.
- Such descriptions can be published in repositories.
- Users can:
  - find service descriptions,
  - compose them into new services, and
  - execute them by referring back to the service providers behind their selection

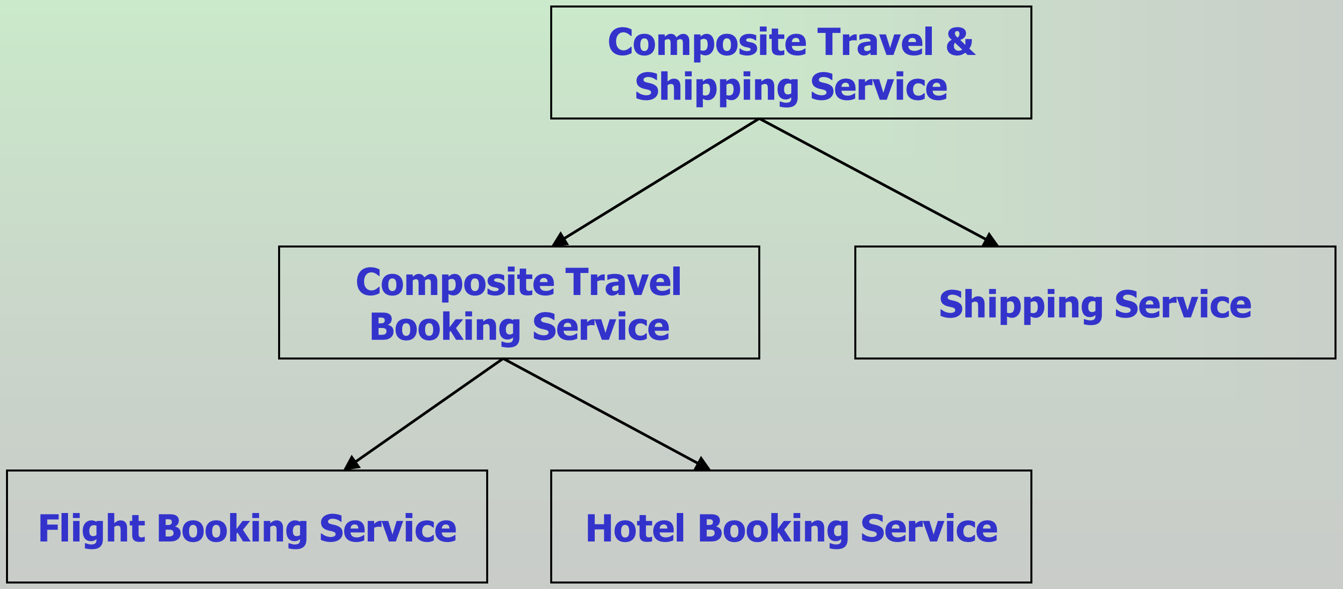
Web Services Composition

- Composition relates to dealing with the assembly of autonomous components so as to deliver a new service out of the existing services.



Hierarchical Composition

- Hierarchical composition refers to the ability to form a composite service by combining already existing services, which themselves might be composed of other composite/primitive services.


Composite Travel & Shipping Service
Composite Travel Booking Service
Shipping Service
Flight Booking Service
Hotel Booking Service

Transactional Composition

- A multi-level model for Web service composition:
  - Hierarchical composition
  - Start with basic activities (services)
    - These are traditional transactions
  - Group them into composite activities
  - Higher level composite activities obtained from lower level basic and/or composite activities
  - Transactional properties extended to composite activities
- Service composition should be treated from a specification and an execution point of view at the same time:
  - The former is about the composition logic
  - The latter is about transactional guarantees

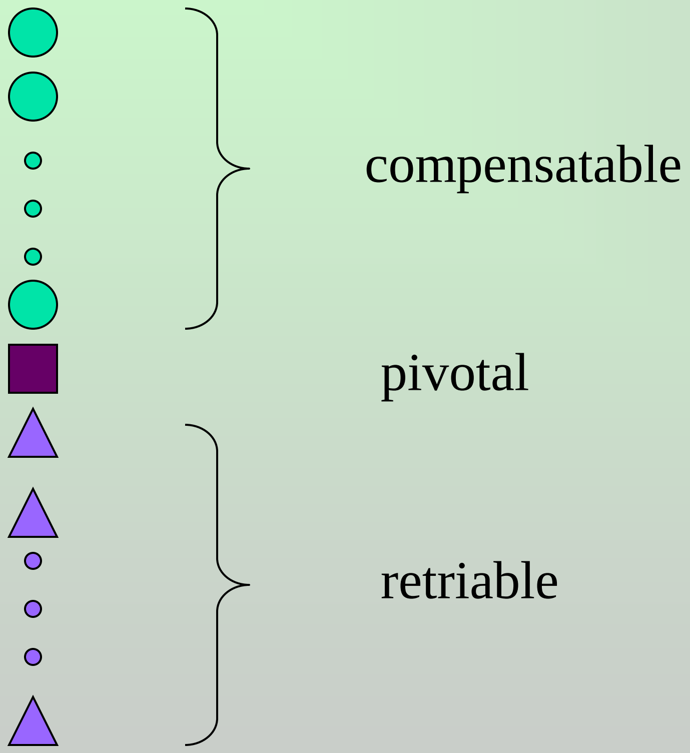
Atomicity

- All or nothing property
- Assumed for traditional transactions and strived for "high-level" transactions
- [Schuldt, Alonso, Beeri and Schek, 2002] extended atomicity properties of multidatabase transactions to transactional processes.
- [Vidyasankar and Vossen, 2004] extended further to composite activities.

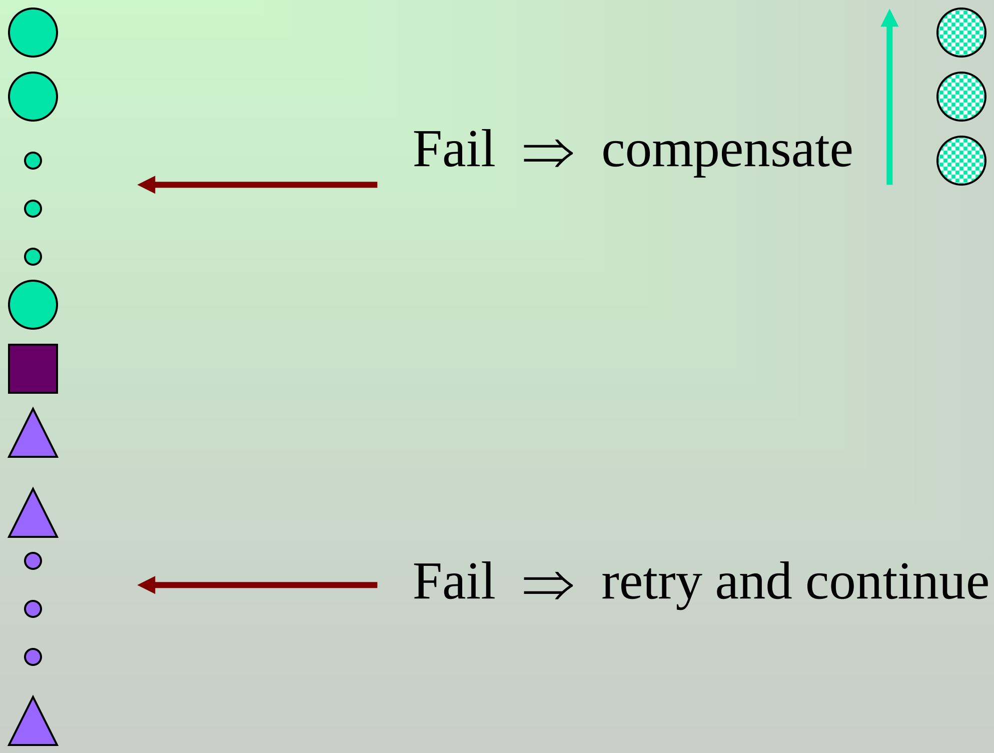
Multidatabase Global Transactions

- Made up of traditional (local) transactions.
- (In a simple form) a sequence of transactions consisting of:
  - A prefix of zero or more compensatable transactions;
  - At most one pivotal (non-compensatable) transaction; and
  - Zero or more retriable (assured) transactions.

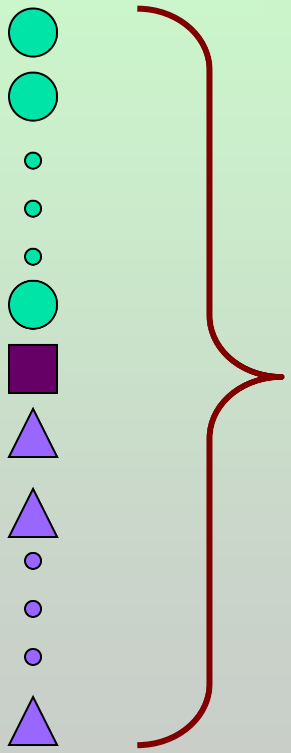
Multidatabase Transaction -1



Multidatabase Transaction - 2



Multidatabase Transaction - 3

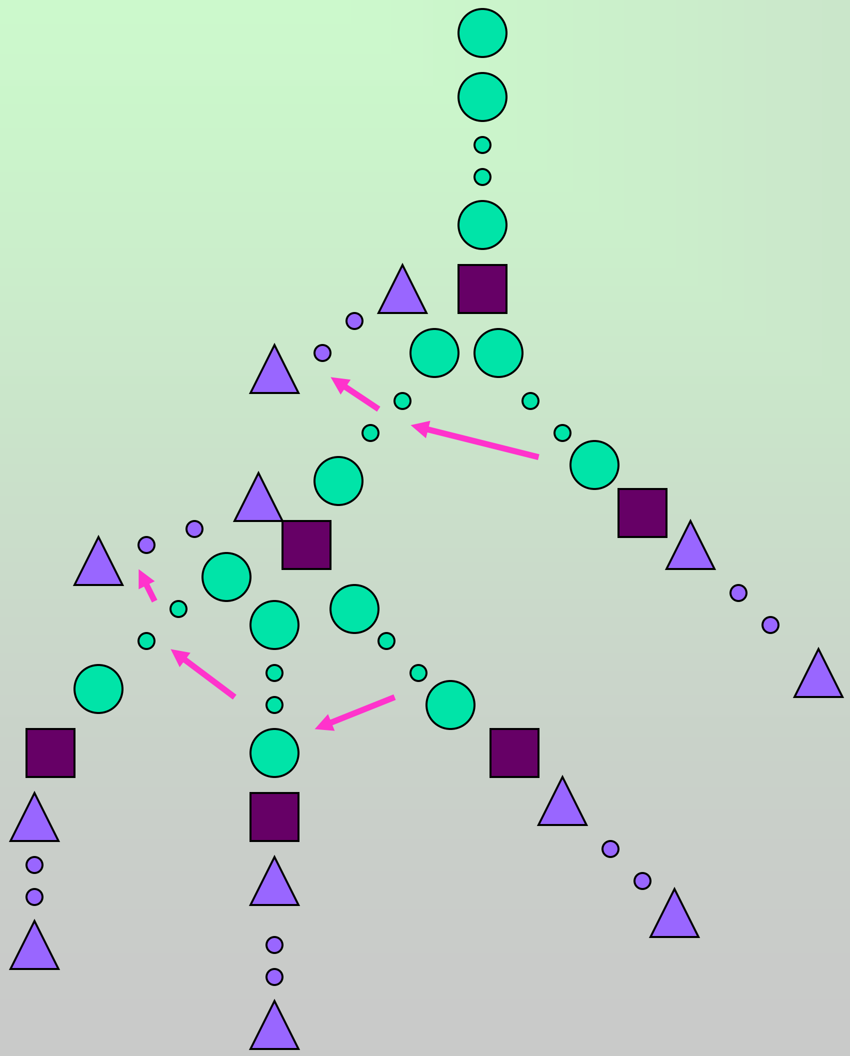


All or nothing property $\Rightarrow$ atomicity

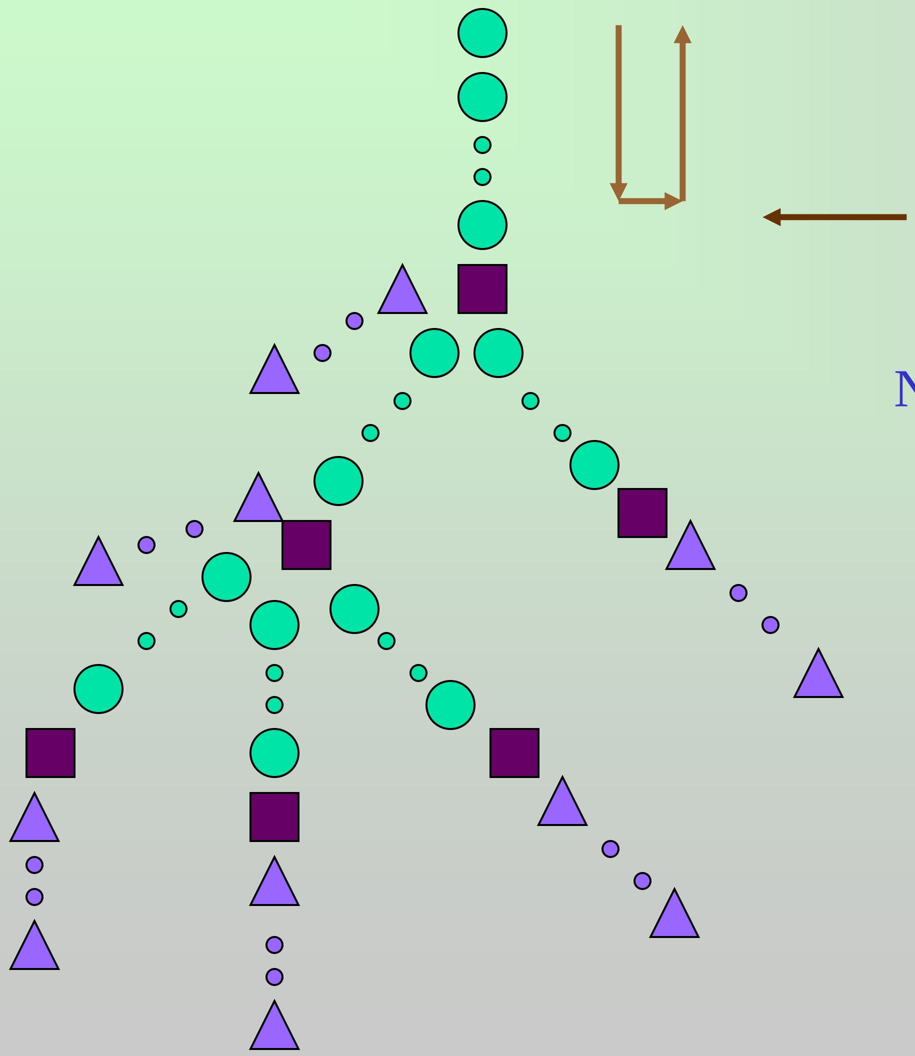
Transactional Processes

- [Schuldt, Alonso, Beeri and Schek, 2002] extended the multidatabase transaction model to transactional processes.
- Composition is a tree.
- Essentially, multiple pivots are accommodated.
- Multiple children are allowed for pivots.
- A preference order is defined on the children.
- The last child is the root of an assured termination tree consisting only of retriable activities.

Process Model Example



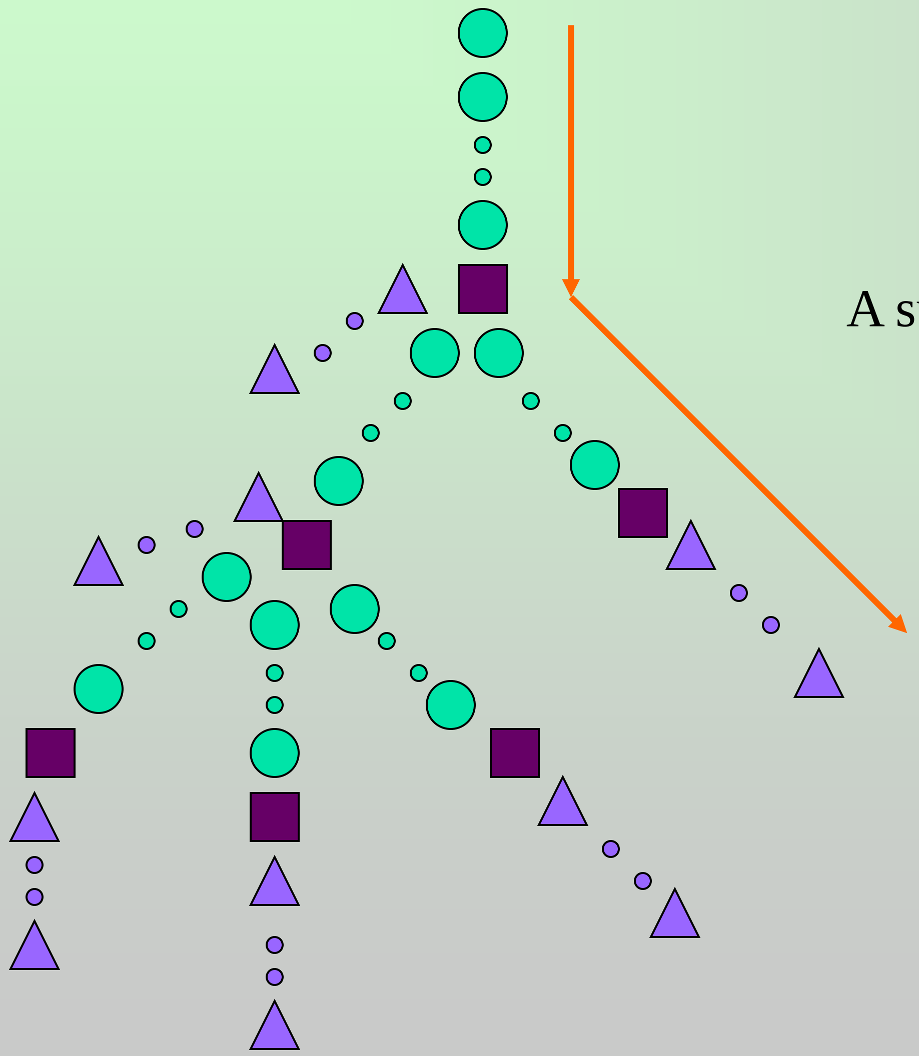
Execution Example - 1



Failure and compensation

Nothing

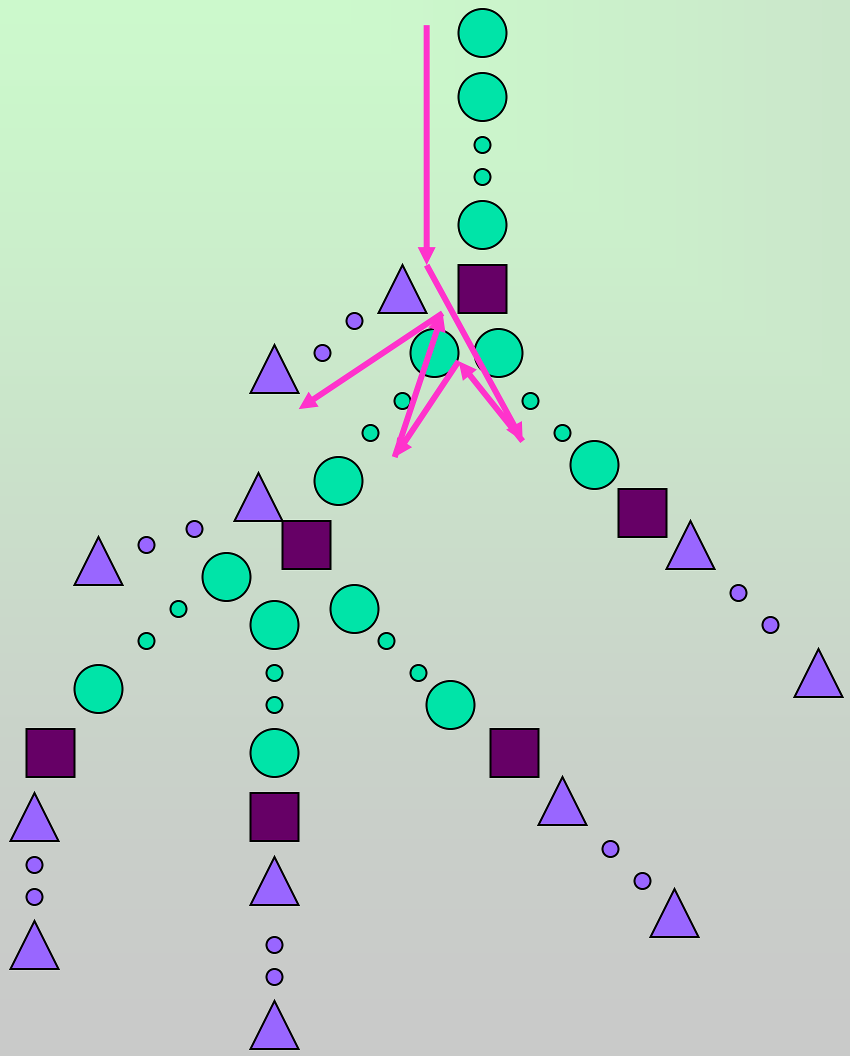
Execution Example - 2



A successful termination

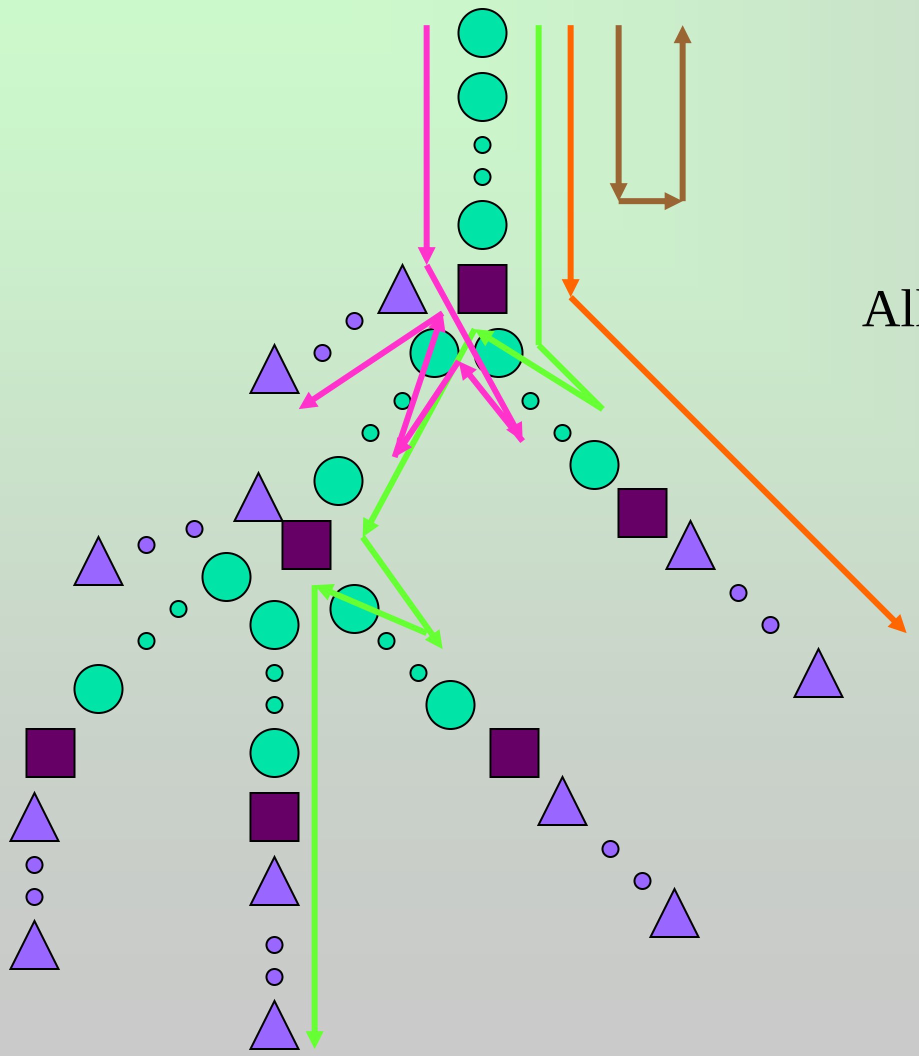
All

Execution Example - 3



Something

Execution Example - 4

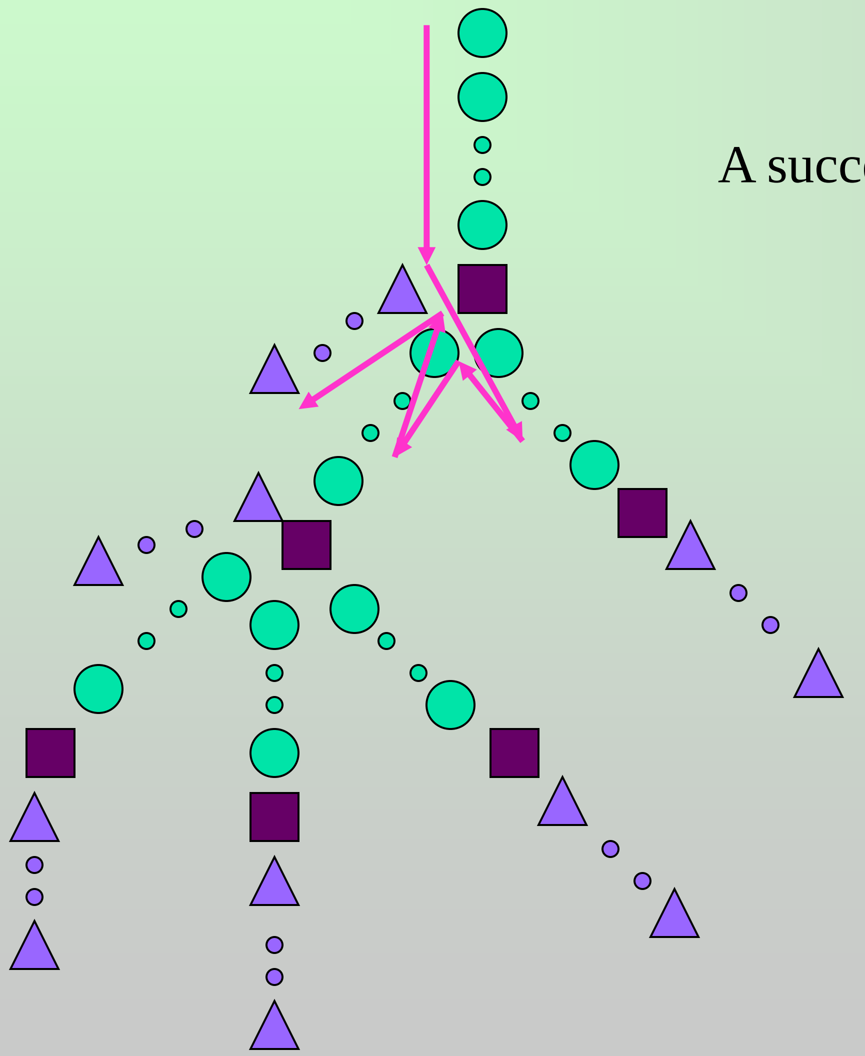


All are guaranteed terminations

Approach

- [Schuldt, Alonso, Beeri and Schek, 2002] extended atomicity of multidatabase transactions to guaranteed termination of transactional processes.
- [Vidyasankar and Vossen, 2004] extend the guaranteed termination property to atomicity (of composite activities).
  - Nothing -- null termination, also failed termination or f-termination.
  - All -- successful termination or s-termination.
  - Something – successful or failed termination. This is relative to the composition, that is, it depends on the application semantics.

Execution Example



A successful or failed termination

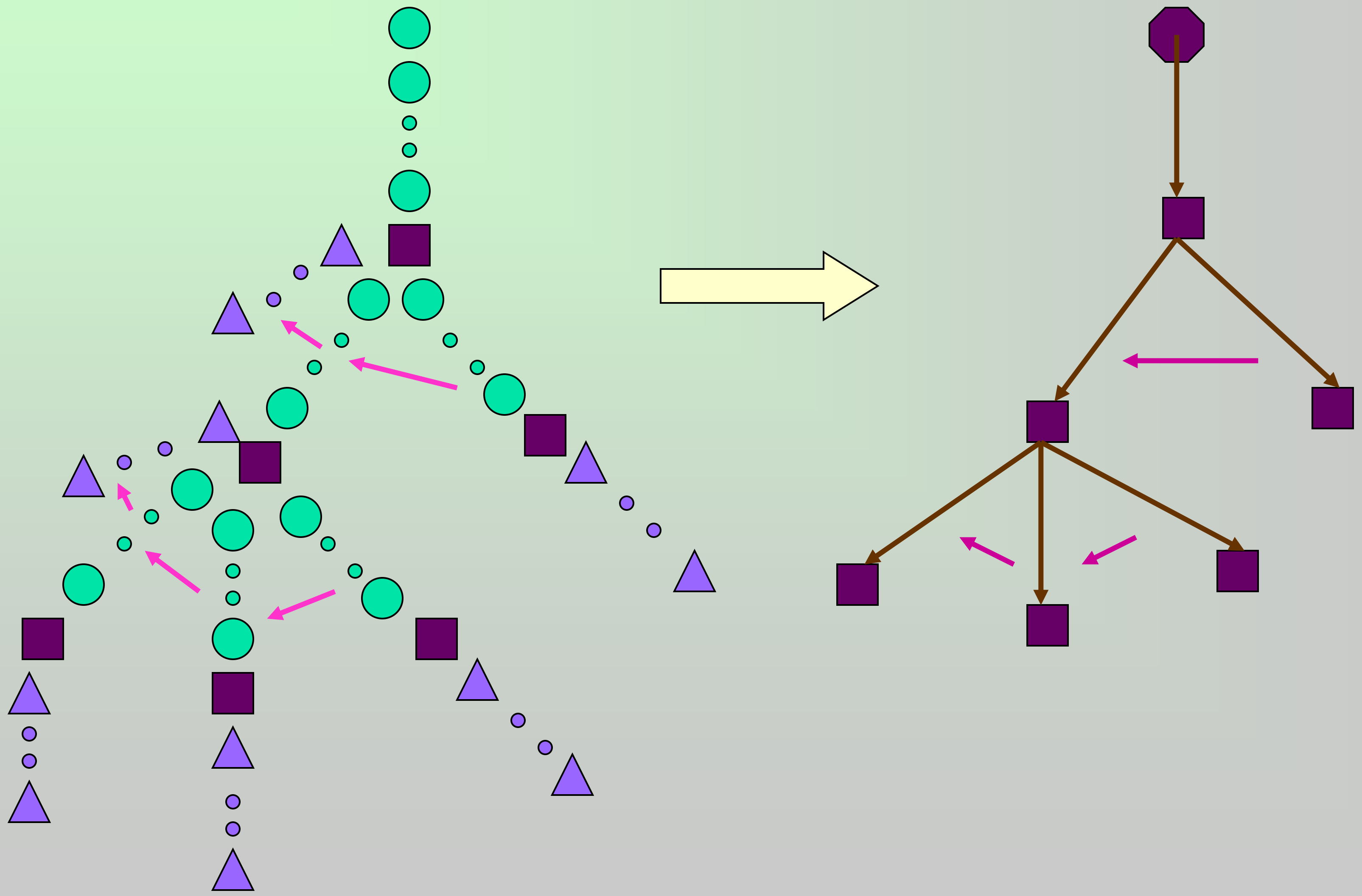
Something

s-termination or f-termination

Pivot Graphs

- To simplify reasoning, a pivot graph, consisting essentially of only the pivotal activities of the composition graph, is defined.
- A dummy pivot is added as the root.
- Then an execution is a path from the root to some node in the graph.

Pivot Graph

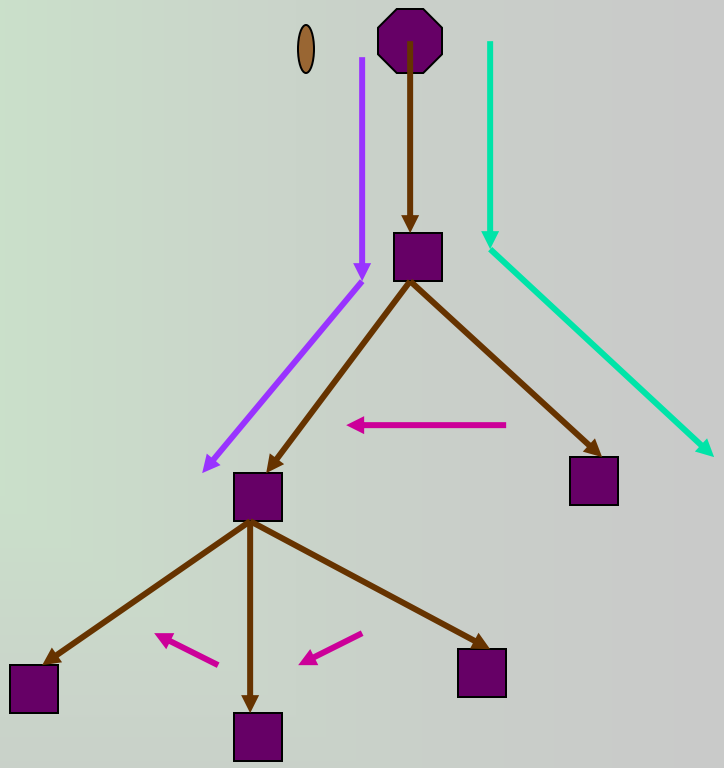


Pivot Graph

Null termination

s-termination

f-termination

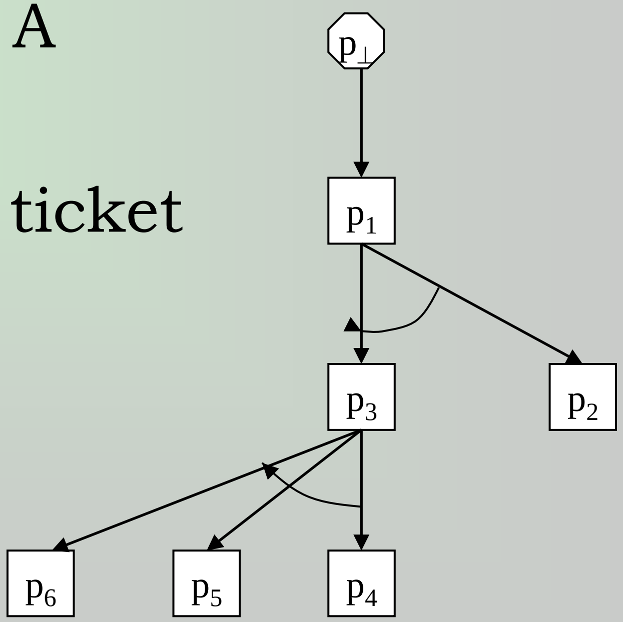


Recoverability

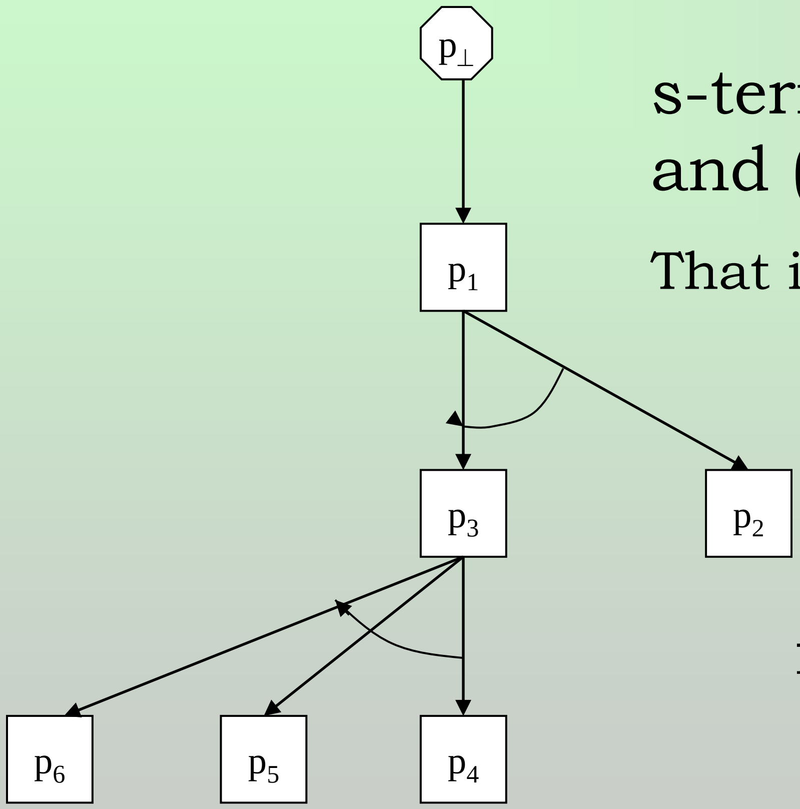
- For achieving atomicity of a composite activity, from an f-termination we should get one of the following:
    - A null termination – by appropriate compensation (at a higher level). This is backward-recoverability.
    - An s-termination – by executing the appropriate suffix (of the composition graph). This is forward-recoverability.

An Application Semantics

- p_1 -- flight ticket purchase
- p_2 -- reservation in conference hotel (A)
- p_3 -- reservation in another (specified) hotel (B)
- p_4 -- shuttle bus from B to A
- p_5 -- car rental
- p_6 -- public transportation ticket



Example

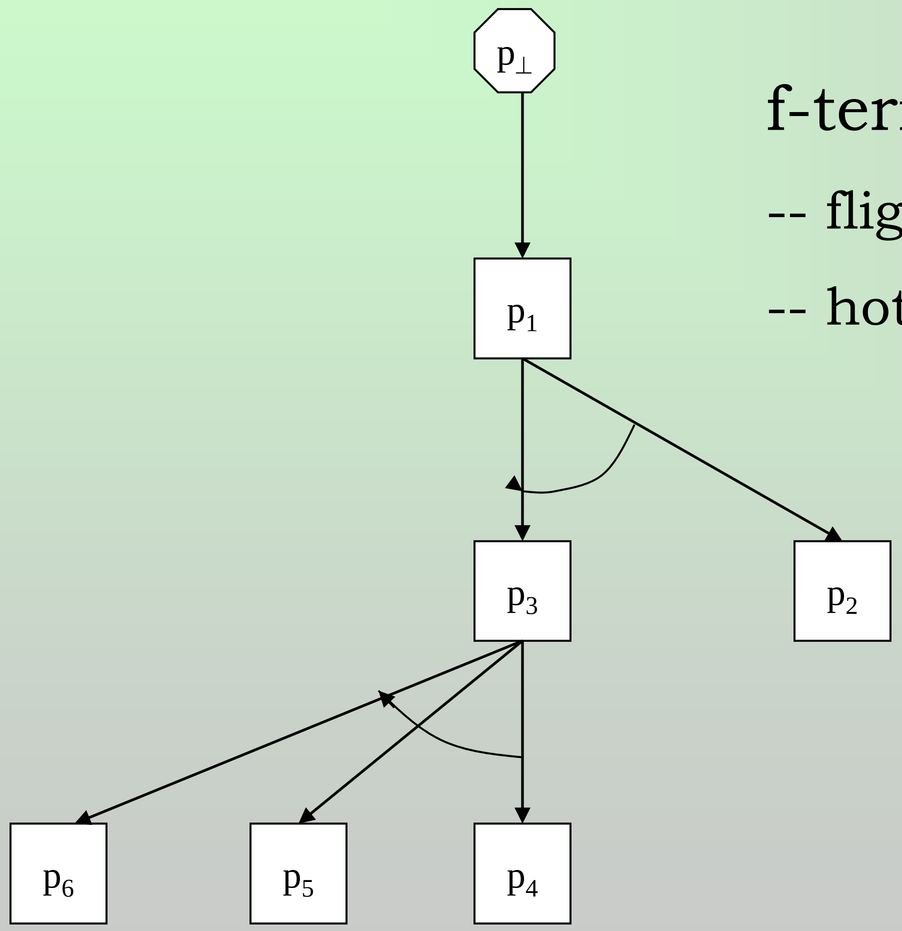


s-terminations: [p_1 and (p_2 or (p_3 and (p_4 or p_5 or p_6)))]

That is, the path from the root to some leaf.

f-terminations: [p_1] and [p_1,p_3]

Example



f-termination $[p_1]$:

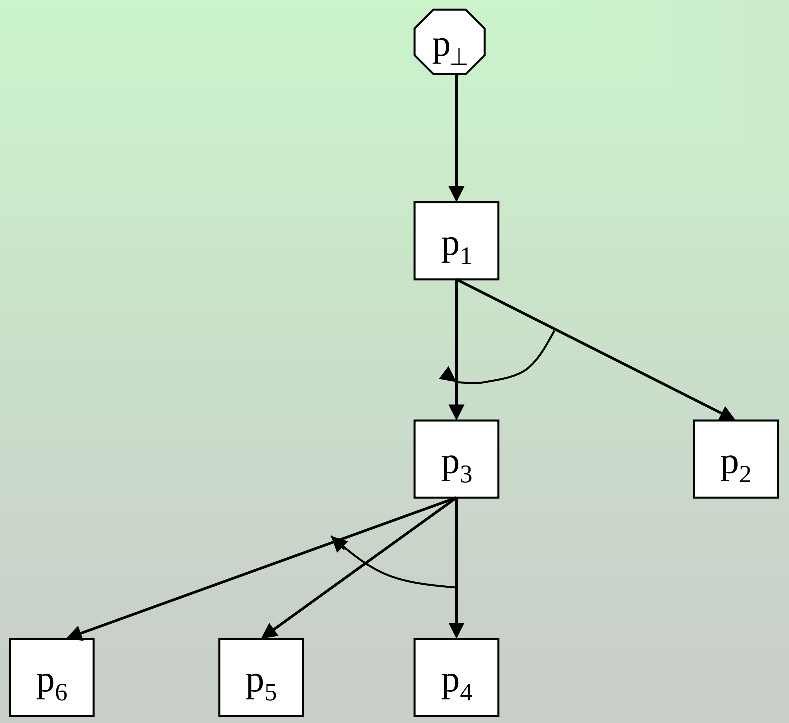
-- flight tickets purchased

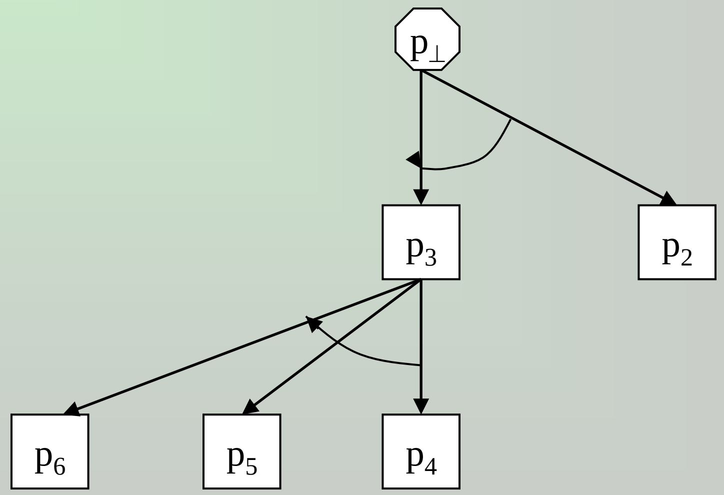
-- hotel reservation not done

Recovery Possibilities

- [p_1] may be compensatable, suffix not retriable
  - Ticket purchase is pivotal at lower level. (Airlines may not refund.)
  - It may be compensatable at higher level. (Travel agency may use the tickets for another customer.)
- [p_1] may not be compensatable, but suffix retriable.
  - Flight tickets cannot be returned.
  - Travel agency does not succeed in hotel reservation.
  - Conference organizers (another service provider) get reservation to the customer directly.

Suffix of [p_1]





Atomic Execution of Composite Activities

- Consider, for example, that a composite activity C is one of the activities in a higher level composition U.
- Atomicity of C is desired in the specification of the composition U.
- Suppose C will be executed by a service provider SC.
- We assume that SC will provide guaranteed termination of C, at the very least.
- Atomicity itself could be the responsibility of SC or of the service requestor SU.
- That is, if SC does not provide atomicity of C, then SU should.

Multi-pivoted Activities

- Consider a composite activity C in a composition U.
- Suppose C is multi-pivoted.
- It may be possible to get an equivalent composition U' where C is replaced by a set of single-pivoted activities.
- For example, suppose C can be replaced by C_1 ;C_2.
- We argue that U may have some added value compared to U'.
- That is, an atomic execution of C by a single service may be more desirable than the atomic executions of the individual sub-activities C_1 and C_2 by different services.

Added Value

- Reduction in the total cost
  - For example, the output of the first activity has to be embedded in an XML document and then extracted by the service provider of the second activity. The document preparation and transportation can be avoided if both activities are executed at the same site.
- Quality of service
  - Implicit dependencies may exist between the two activities affecting the quality of service if executed in different sites.
- Atomicity guarantee
  - C_1 may not be compensatable and C_2 not retriable, but a service provider can keep C_1 in a prepared-to-commit state until the execution of C_2 reaches the commit stage and commit both of them together.
- Increased security and autonomy
  - Not letting out trade, contract, or service secrets.

Transactional Properties for Composite Activities

- Once we have the notion of atomicity for a composite activity, we can talk about compensatability, pivotal, and retriable properties also.
  - These will be relative to the composition.
- In higher level compositions, basic and composite activities are composed into a higher level activity.
- From the atomicity and the c, p, r properties of the constituent activities, we can define the atomicity (and other properties) of the higher level activity.
- This can be carried out to any level.

Electronic Contracts

Vidyasankar, Radha Krishna and Kamalakar Karlapalem,
2007, 2008, 2009

[VRK] References

- K. Vidyasankar, P. Radha Krishna and Kamalakar Karlapalem [VRK]:
  - A Multi-level Model for Activity Commitments in E-contracts, 15th Int. Conf. in Cooperative Information Systems (CoopIS 2007), Vilamora, Portugal, LNCS Vol. 4803, Springer 2007, pp. 300-317.
  - Study of Execution Centric Payment Issues in E-contracts, Proc. IEEE Services Computing Conference (SCC), Hawaii, July 2008, Vol. 2, pp. 135-142.
  - Study of Dependencies in Executions of E-contract Activities, 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009), Riga, Latvia, September 2009, LNCS Vol. 5739, Springer 2009, pp. 301-313.

Electronic contract (e-contract)

- An e-contract is a contract modeled, specified, executed, controlled and monitored by a software system.
- A contract is a legal agreement involving parties, activities, clauses and payments.
- The activities are to be executed by parties satisfying clauses, with the associated terms of payment.

Example: Contract for building a house

- Parties: Customer, builder, bank, insurance company.
- The builder constructs house as per the customer's specifications; some activities such as plumbing and electrical work may be sub-contracted.
- The customer gets mortgage from the bank.
- The house is insured comprehensively for the market value covering fire, flood, etc. in the joint names of the bank and the customer.
- Several bi-lateral or tri-lateral contracts may exist for building the house. We consider all of them to be part of a single high-level contract.

Complexity of contracts

- Contracts are complex in nature.
- Both the initial specification of the requirements and the later verification of the execution with respect to compliance to the clauses are very tedious and complicated.
- This is, partly, due to the complexity of activities.
  - Activities may be electronic or non-electronic.
  - They are interdependent with other activities and clauses.
  - They may be executed by different parties autonomously, in a loosely coupled fashion.
  - They are long-lasting.
  - The outcomes of their executions may be unpredictable.

Goals of the e-contract

- The premise is that, to handle the complexity of a contract, an e-contract should reflect both the specification and the execution aspects of the activities at the same time, where the former is about the composition logic and the latter is about the transactional properties.
- Hence, the goals of an e-contract include:
  - precise specification of the activities;
  - mapping them into deployable workflows;
  - and providing transactional support in their execution.

Properties of activities in e-contracts

- Compensatability and retriability are encountered in the execution of e-contract activities also, that too in sophisticated ways:
  - Both complete and partial executions may be compensated;
  - Both successful and unsuccessful executions may be compensated;
  - Even "committed" executions may be retried;
  - Retrying may mean, in addition to re-execution, "adjusting" the previous execution; and
  - Activities may be compensated and/or retried at different times, relative to the executions of other activities.

Some examples

- (Time of compensation) An issued cheque can be cancelled only if it has not been cashed already.
- (Adjusting the execution) In the process of repayment of a bank loan, if a cheque is bounced for some reason, the customer has to pay a penalty in addition to the actual amount.

Closure and E-contract Commitment

- Each activity must be closed at some time. On closure, no execution related to that activity would take place.
- The closure could be done on a complete or incomplete execution, and on a successful or failed execution.
- On closure of the contract-activity, the e-contract itself can be closed. (This may involve settlement of payment and other issues between the parties.)
- E-contract closure is also referred to as e-contract commitment.
- The term e-contract commitment logic is used to refer to the entire logic behind the commitment of the various activities of the e-contract, and the closure of the activities and the e-contract.

Multi-Level Composition Model

- [Vidyasankar, Radha Krishna and Kamalakar Karlapalem, 2007] propose a framework for e-contract commitment.
- The multi-level Web service composition model is extended to e-contract activities.
- Transactional properties are defined for the activities in every level. These properties include:
  - successful termination;
  - Compensatability;
  - Retriability;
  - forward and backward recoveries; and
  - commitment.
- This is done uniformly, the same way irrespective of the level of the activity.

Properties of e-contract activities

- E-contract activities differ from database transactions in many ways:
  - (i) Different successful executions are possible for an activity;
  - (ii) Unsuccessful executions may be compensated or re-executed to get different results;
  - (iii) Whether an execution is successful or not may not be known until after several subsequent activities are executed, and so it may be compensated and/or re-executed at different times relative to the execution of other activities;
  - (iv) Compensation or re-execution of an activity may require compensation or re-execution of several other activities; etc.

Basic activities

- Some activities are considered as basic.
- These cannot be decomposed into smaller ones, or we want to consider them in entirety.
- They may be electronic (e.g., processing a payment) or non-electronic (e.g., painting a door).
- We would like their execution to be atomic, that is, either not executed at all or executed completely.
- However, incomplete executions are unavoidable and we consider them also.

Constraints

- Each activity is executed under some constraints
  - Who can execute, when can it be executed, which executions are acceptable, etc.
- A complete or incomplete execution satisfying the constraints specified at the time of the execution is called a successful termination (s-termination).
- The constraints are specified in terms of an s-termination-predicate (st-predicate).
- An execution which does not satisfy the st-predicate is a failed termination (f-termination).

Example – Painting a wall

- The execution is
  - Incomplete while being painted.
  - Complete after the painting is finished.
  - s-termination if the paint job satisfies the st-predicate:
    - One undercoat and one other coat; and
    - no smudges in the ceiling or adjacent walls.
  - f-termination otherwise.

Change of constraints

- Constraints may change, that is, the st-predicate of an activity may change, as the execution of the contract proceeds.
  - In the example of painting a wall, the requirement of one coat (in addition to one undercoat) may be changed to two coats.
- Such changes may invalidate a previous execution. Then, the execution needs to be adjusted.

One way of adjusting - Compensation

- Compensation is to nullify the effects of the execution. Options are:
  - absolute compensation if possible;
  - ignoring the original execution;
  - executing a compensating activity; etc.
- Compensation may be constrained by time.
  - Example: Purchased goods cannot be returned after 7 days.
- For an activity, some (not necessarily all) executions may be compensatable.
  - Flight tickets may be fully refundable, partially refundable or non-refundable.
  - Which tickets will be available may not be known in advance.
- Therefore, compensatability property is attributed to an execution of the activity, not to the activity itself.

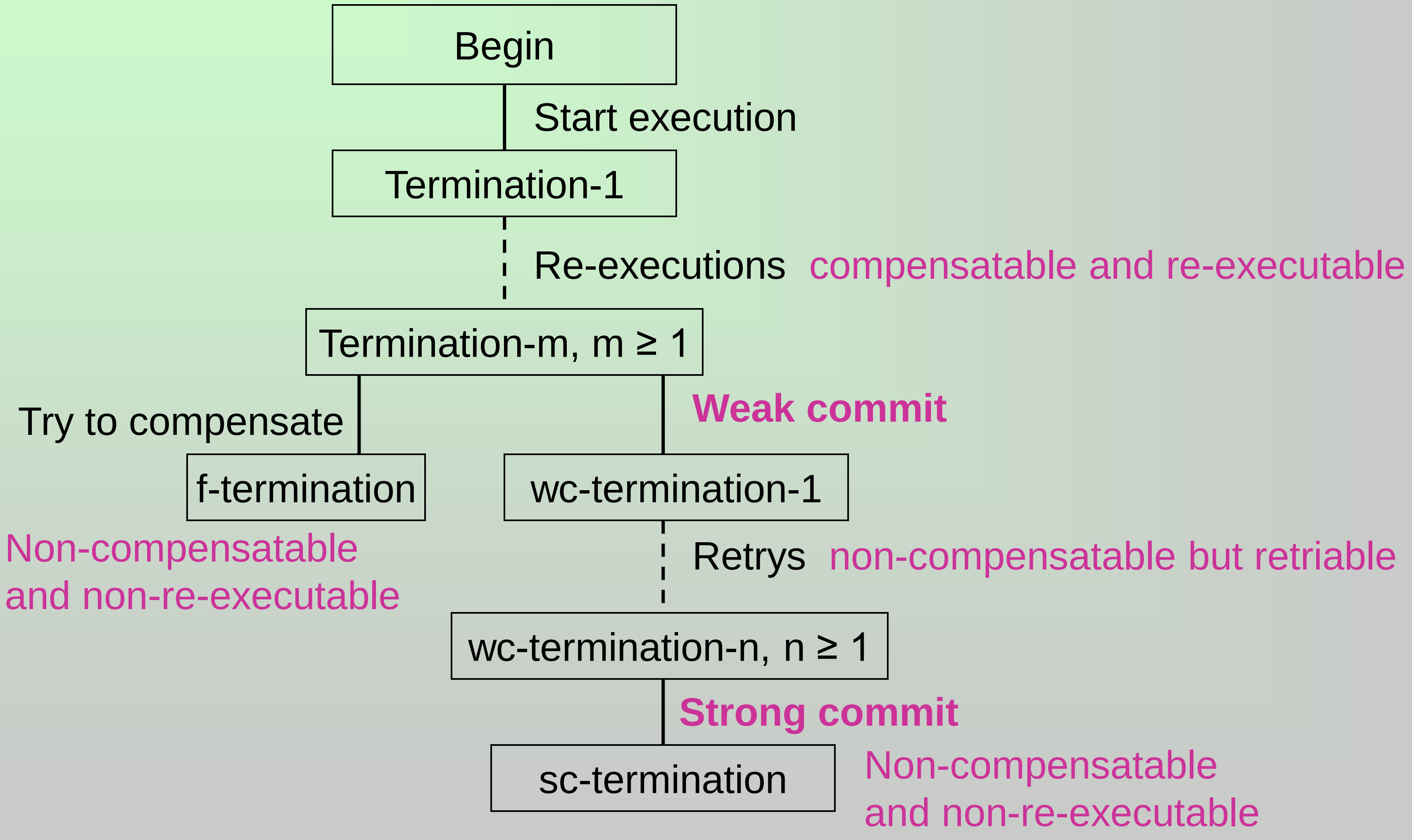
Another way of adjusting - Retry

- Retriability is the ability to get a complete execution satisfying the (possibly new) st-predicate by re-executions.
  - Retrying may involve a partial or full roll back and then a re-execution.
  - Retriability may also be time-dependent.
- Some executions of an activity may be retriable, some others may not be retriable.
- Again, retriability is attributed to an execution of the activity, not to the activity itself.
- Retriability property is orthogonal to compensatability.

Execution states of an activity

- We consider an execution of an activity with a specified st-predicate.
- On a termination, if we are not satisfied with the outcome, we may re-execute.
- Several re-executions and terminations are possible.
- We assume the following progression of the states of the (complete or incomplete) terminations.

Execution states of an activity

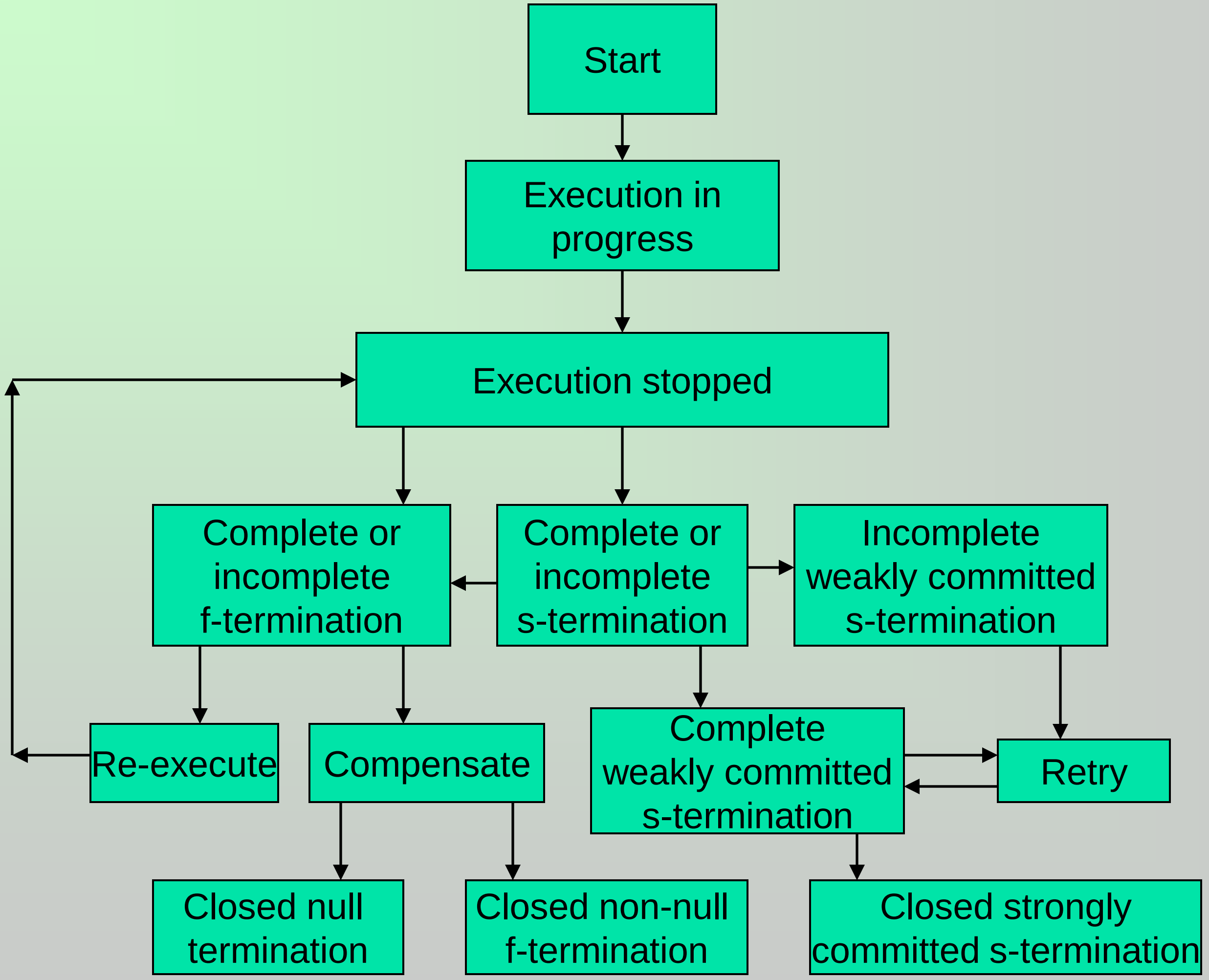


Progression of states

1. The termination is both compensatable and re-executable.
2. At some stage, the termination becomes non-compensatable, but is still re-executable. Then, perhaps after a few more re-executions, we get a termination which is either
   - (a) non-re-executable to get a complete s-termination (we take this as a f-termination), or
   - (b) re-executable to get eventually a complete s-termination. We identify this state as non-compensatable but retriable. The execution in this state is said to be weakly committed.

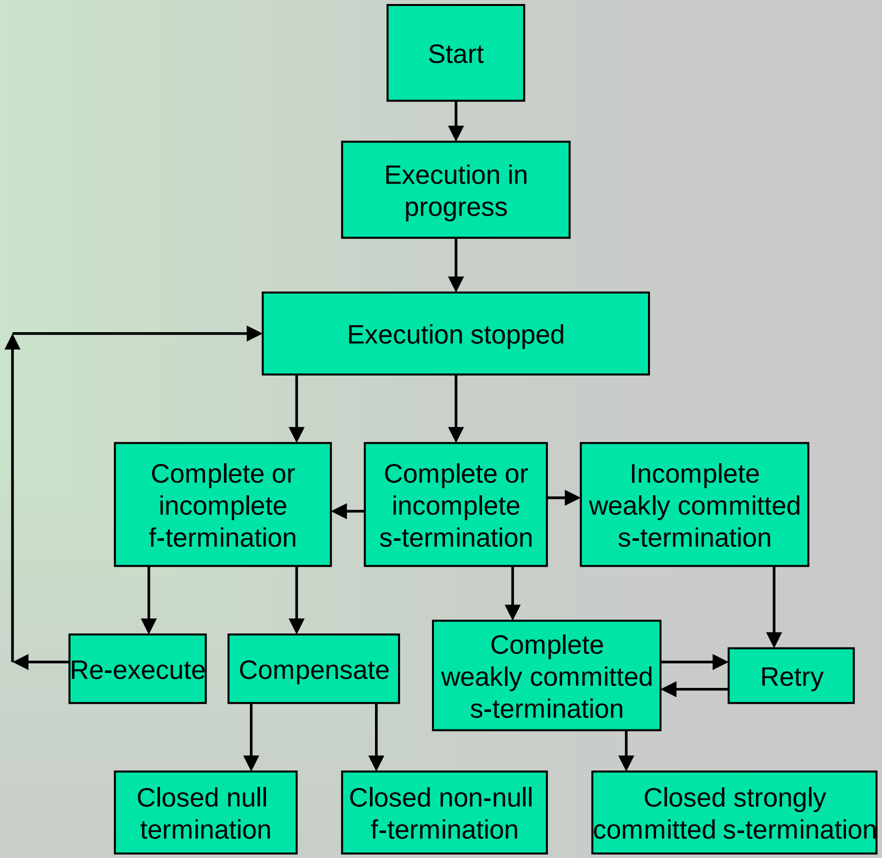
- Continuing re-executions in state 2.(b), at some stage, we get a complete s-termination which is non-compensatable and non-re-executable. Here the execution is said to be strongly committed.

Execution stages of an activity


Start
Execution in progress
Execution stopped
Complete or incomplete f-termination
Complete or incomplete s-termination
Incomplete weakly committed s-termination
Re-execute
Compensate
Complete weakly committed s-termination
Retry
Closed null termination
Closed non-null f-termination
Closed strongly committed s-termination

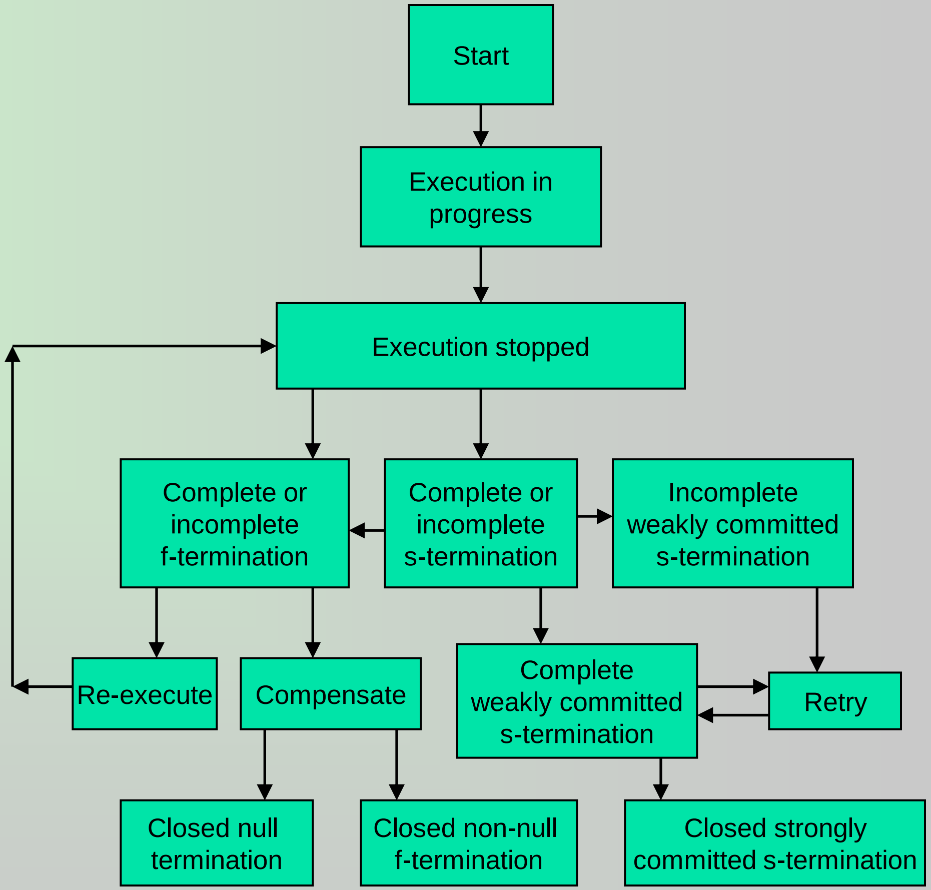
Some points

- Retrys and re-executions are possibly after partial or full backward recovery.
- A complete s-termination may become f-termination, with a change in st-predicate.
  - If this happens before weak commitment, the transitions of an f-termination are followed.
  - If the execution is already weakly committed, then a retry that guarantees s-termination is assured.
- If the compensation succeeds, we get the null termination. Otherwise, we get a non-null f-termination.


Start
Execution in progress
Execution stopped
Complete or incomplete f-termination
Complete or incomplete s-termination
Incomplete weakly committed s-termination
Re-execute
Compensate
Complete weakly committed s-termination
Retry
Closed null termination
Closed non-null f-termination
Closed strongly committed s-termination

Additional points

- The "final" state of execution is closure.
- Three possible states of closure are shown:
  - Null;
  - Non-null (complete or incomplete) f-termination; and
  - Complete s-termination, which also corresponds to strong commitment of the execution.



Start
Execution in progress
Execution stopped
Complete or incomplete f-termination
Complete or incomplete s-termination
Incomplete weakly committed s-termination
Re-execute
Compensate
Complete weakly committed s-termination
Retry
Closed null termination
Closed non-null f-termination
Closed strongly committed s-termination

Hierarchical composition

- Our hierarchical composition of the activities is:
  - In the first level, a composite activity consists of basic activities;
  - In the next level, a composite activity consists of basic and/or composite activities of level one; etc.
  - The highest level will have the "single" activity for which the contract is made. We call this the contract-activity.
  - There could be multiple contracts for a single activity. For building a house, there could be separate contracts between (i) customer and the bank, (ii) customer and the builder, and (iii) the builder and the bank. We consider this set of contracts as a part of a single high level contract whose contract-activity is building a house.

Composition graph – Bottom level

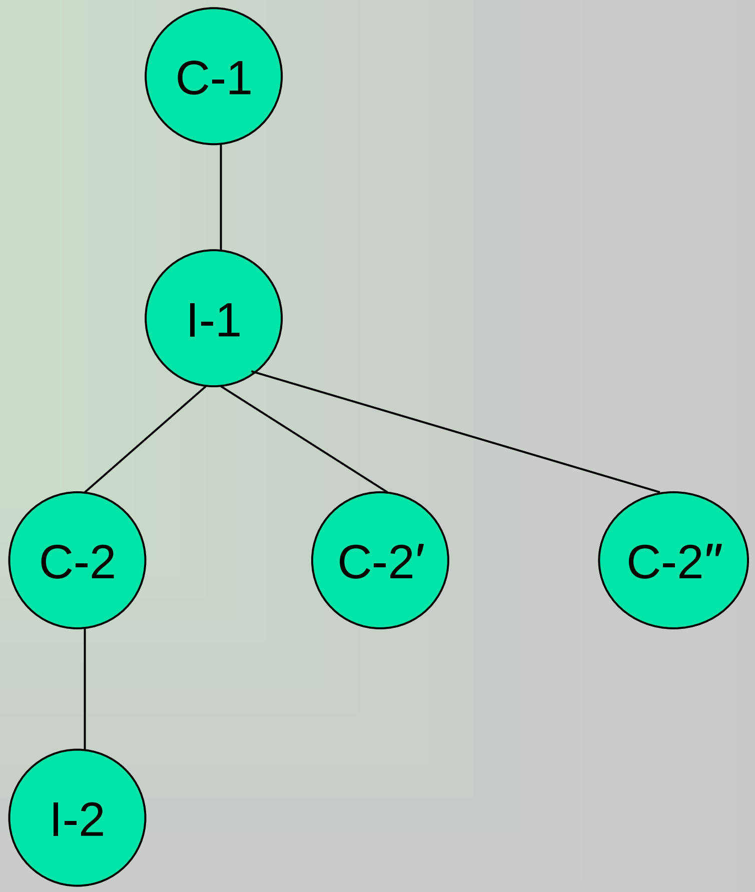
- Composition **C** is a rooted tree. It is a part of a higher level composition **U**.
- Nodes in the tree correspond to basic activities.
- With each node, an st-predicate which specifies the s-terminations of that activity is prescribed.
- A children execution predicate (ce-predicate) is also associated with each node. This specifies, for each s-termination of that node, a set of children which have to be executed.

Composite activity

- An execution E of **C** yields a composite activity C. It consists of executions of activities in the paths from the root to some leaves. This is called the execution-tree of E.
- If all the activities in these paths have been executed completely, then E is a complete execution of **C**.
- Otherwise, if only the activities from the root to some non-leaf nodes have been executed and/or the executions of some activities are not complete, then it is an incomplete execution of **C**.

Composition example

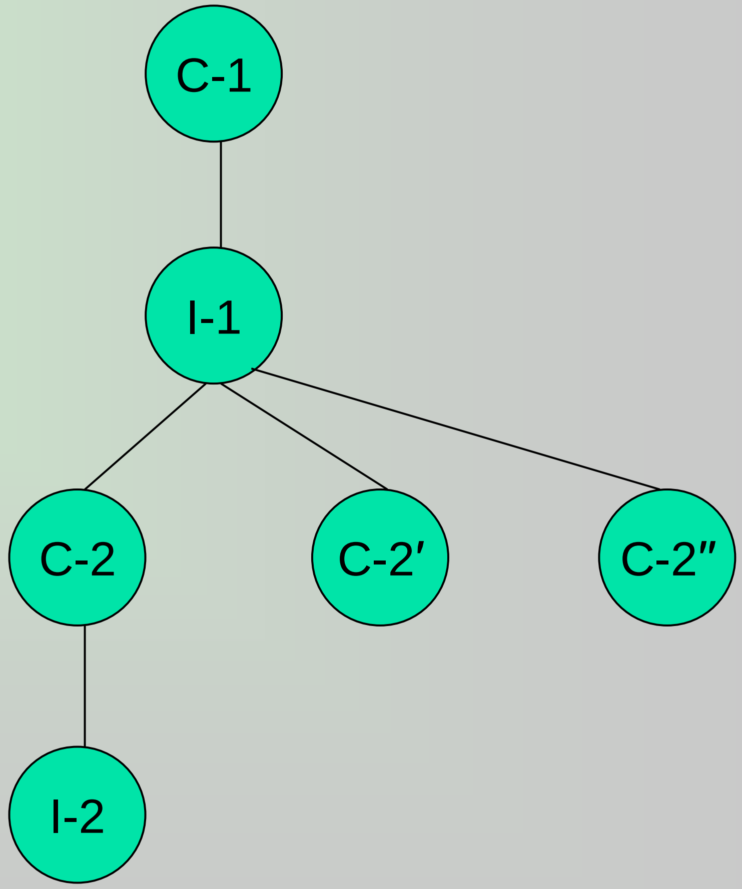
- Construction activities C-i for a product, and inspection activities I-i.
- The st-predicate for each C-i will be the guidelines for that step. The st-predicate for each I-i will be the acceptable results of that inspection.
- After C-1, I-1 is carried out. Depending on the result, C-2 is to be carried out if possible, and either C-2′ or C-2″ otherwise. This is the ce-predicate at I-1.



A composition

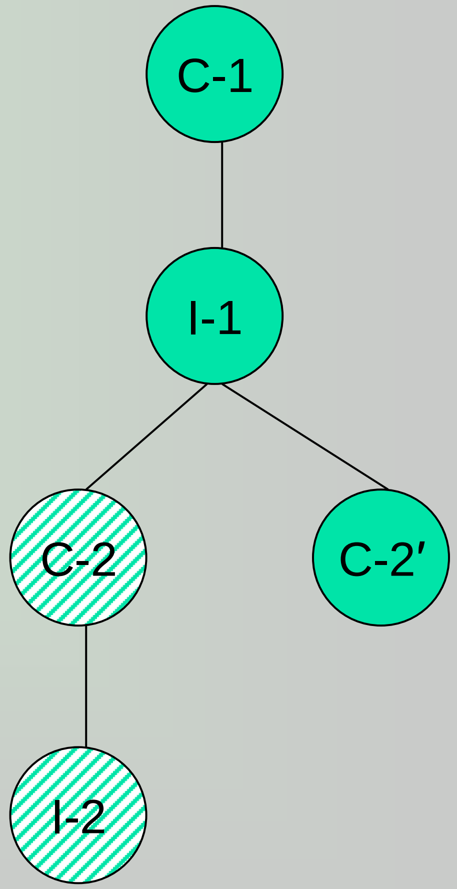
Execution example - 1

- Suppose C-2 was executed after I-1, and I-2 fails.
- It may be decided that the product be sent back to C-1 for some fixing, inspected again, and then the options C-2′ and C-2″ explored.
- This amounts to compensating I-2 and C-2, and retrying C-1 and I-1, each possibly with adjusted st-predicates. The adjusted ce-predicate for I-1 will have only C-2′ and C-2″ options.



Execution example – 2

- Suppose C-2′ is tried and the execution was successful.
- Then the execution-tree is as shown. Here C-2 and I-2 are f-terminations.



Composite activity - Terminations

- If each activity in E has s-terminated, then E is a (complete or incomplete) s-termination of **C**.
- In a (complete or incomplete) f-termination, executions of some activities have f-terminated.
- The execution of each s-terminated node satisfies the st-predicate of that node
- In a complete s-termination, the selection of children at each non-leaf node satisfies the ce-predicate at that node.
- Both st- and ce-predicates of the nodes may change as the execution of **C** proceeds.

Commitment of constituent activities

- Execution of each activity in C may first be weakly committed; then it is strongly committed, some time after its s-termination.
- Once weakly committed, the execution cannot be compensated; and once strongly committed, it cannot be retried.
- The activities in C are (both weakly and strongly) committed in sequence. That is, when an activity is weakly committed, all preceding activities in C are also weakly committed. The same holds for strong commitment.

Transactional properties

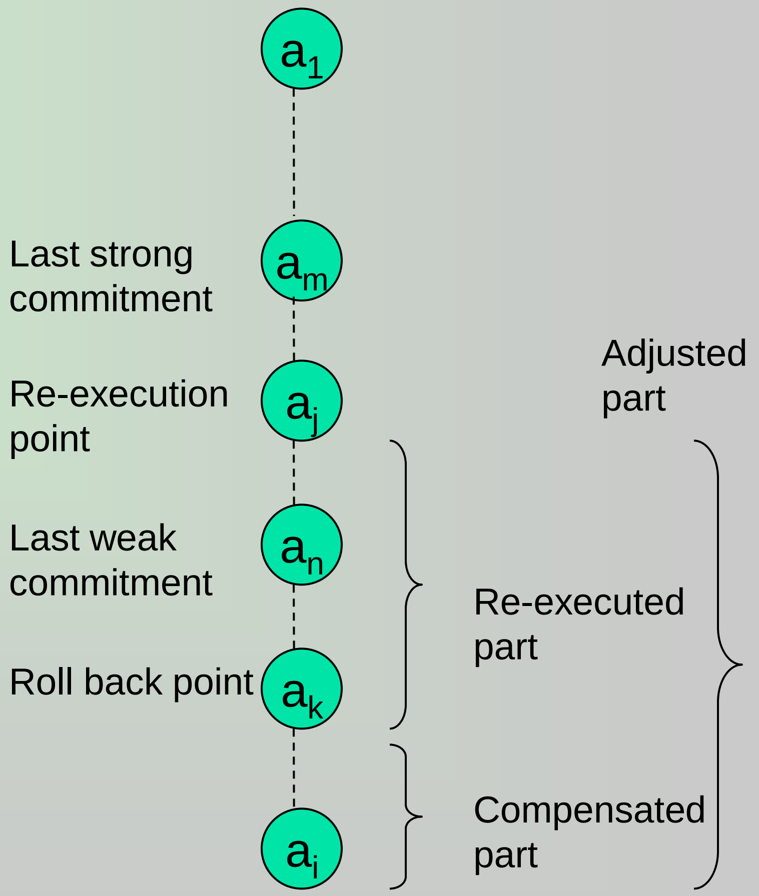
- Weak commitment, strong commitment, compensatability and retriability of the activities in **C** are all relative to **C**. (We explain this shortly.)
- Composition **C** assumes that each of its activities is executed atomically. Then, an f-termination is assumed to be compensatable, relative to **C**.

Transactional properties (cont’d)

- The execution of the entire composition **C** is intended to be atomic in the higher level composition **U**.
- If E is an incomplete s-termination, forward recovery is carried out by executing the “suffix” of E in **C**, to get a complete s-termination.
- If E is an f-termination, then the executions of some activities may have to be adjusted (partial backward recovery) to get an incomplete s-termination, and then a forward recovery is carried out.
- To get the null termination, E has to be compensated (full backward recovery).

Partial backward recovery – Simple case

- f-termination of a_i may warrant adjustment of executions:
  - Re-execution of a_j;
  - If it does not succeed, a_j and all executions up to a_i are compensated.
  - If the re-execution succeeds with an s-termination and the ce-predicate corresponding to that s-termination allows re-execution of a_{j+1}, then re-execute a_{j+1}. Otherwise, compensate a_{j+1} and all executions up to a_i; and so on.



Example - Execution of Activities and Terms of Payments

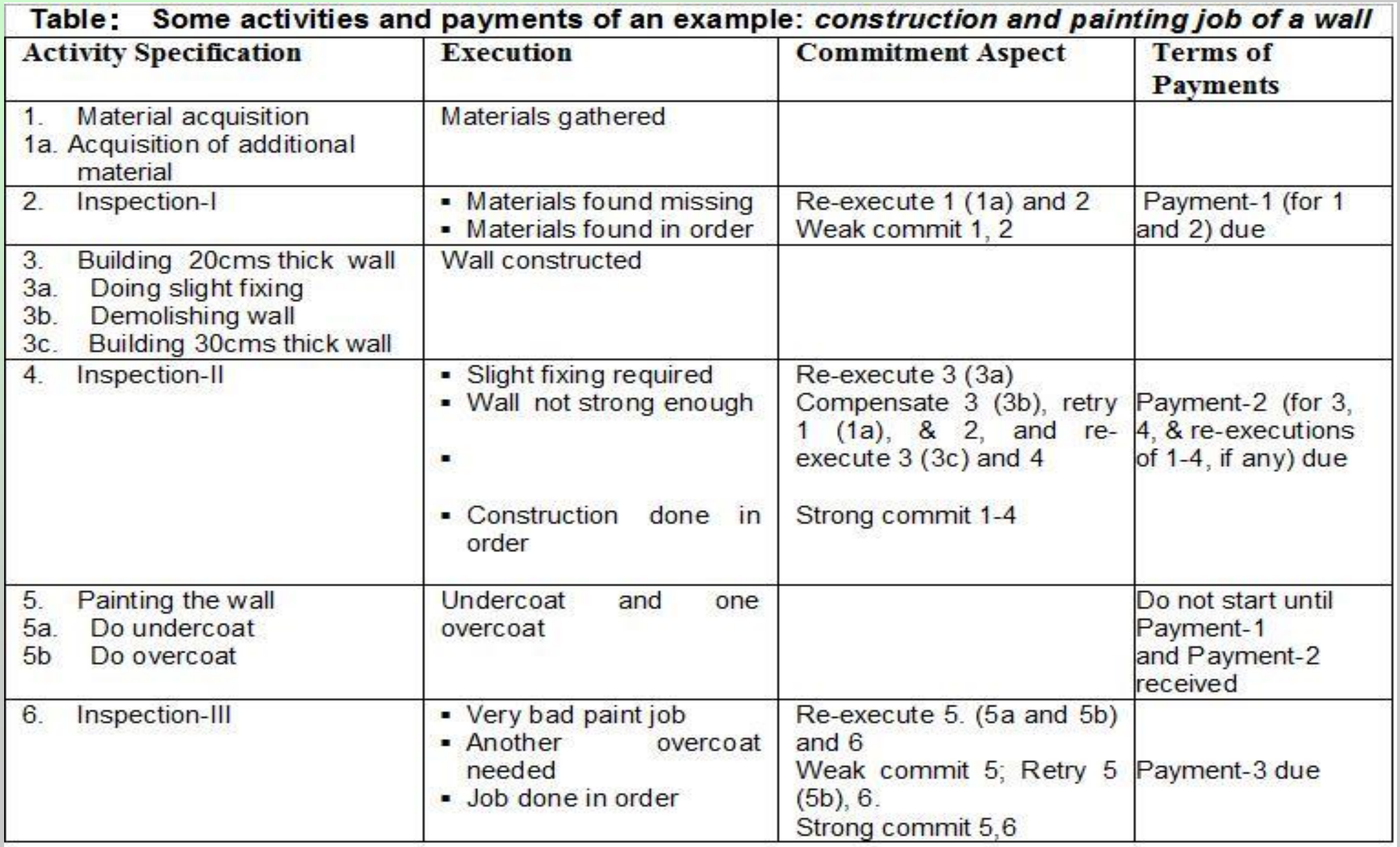


Table: Some activities and payments of an example: *construction and painting job of a wall*

Activity Specification	**Execution**	**Commitment Aspect**	**Terms of Payments**
1. Material acquisition 1a. Acquisition of additional material	Materials gathered		
2. Inspection-I	▪ Materials found missing ▪ Materials found in order	Re-execute 1 (1a) and 2 Weak commit 1, 2	Payment-1 (for 1 and 2) due
3. Building 20cms thick wall 3a. Doing slight fixing 3b. Demolishing wall 3c. Building 30cms thick wall	Wall constructed		
4. Inspection-II	▪ Slight fixing required ▪ Wall not strong enough ▪ ▪ Construction done in order	Re-execute 3 (3a) Compensate 3 (3b), retry 1 (1a), & 2, and re-execute 3 (3c) and 4 Strong commit 1-4	Payment-2 (for 3, 4, & re-executions of 1-4, if any) due
5. Painting the wall 5a. Do undercoat 5b Do overcoat	Undercoat and one overcoat		Do not start until Payment-1 and Payment-2 received
6. Inspection-III	▪ Very bad paint job ▪ Another overcoat needed ▪ Job done in order	Re-execute 5. (5a and 5b) and 6 Weak commit 5; Retry 5 (5b), 6. Strong commit 5,6	Payment-3 due

Example - Execution of Activities and Terms of Payments

- House-Building Contract: Construction & Painting Job of a Wall
  - Work begins with gathering of all required materials such as bricks, cement, paint, paint brushes, etc.
  - On Inspection-I, if some materials are missing, then they are also gathered (re-execution of activity 1).
  - A 20cms thick wall is constructed as per the building specifications.
  - Then Inspection-II is done to check the strength of the wall and the quality of the job done.
    - If slight fixing is needed, some more work is done (re-execution) and then the inspection is carried out again.
    - If (zero or more) slight fixes do not yield satisfactory results, the wall is demolished (compensating activity)
    - A 30cms thick wall is built, starting from acquisition of further material. (This is a partial roll back of the execution.)
    - Even with the new wall, slight fixing and/or complete demolishing and re-building may occur, in fact, several times.
  - When the wall is constructed to the satisfaction, it is painted.

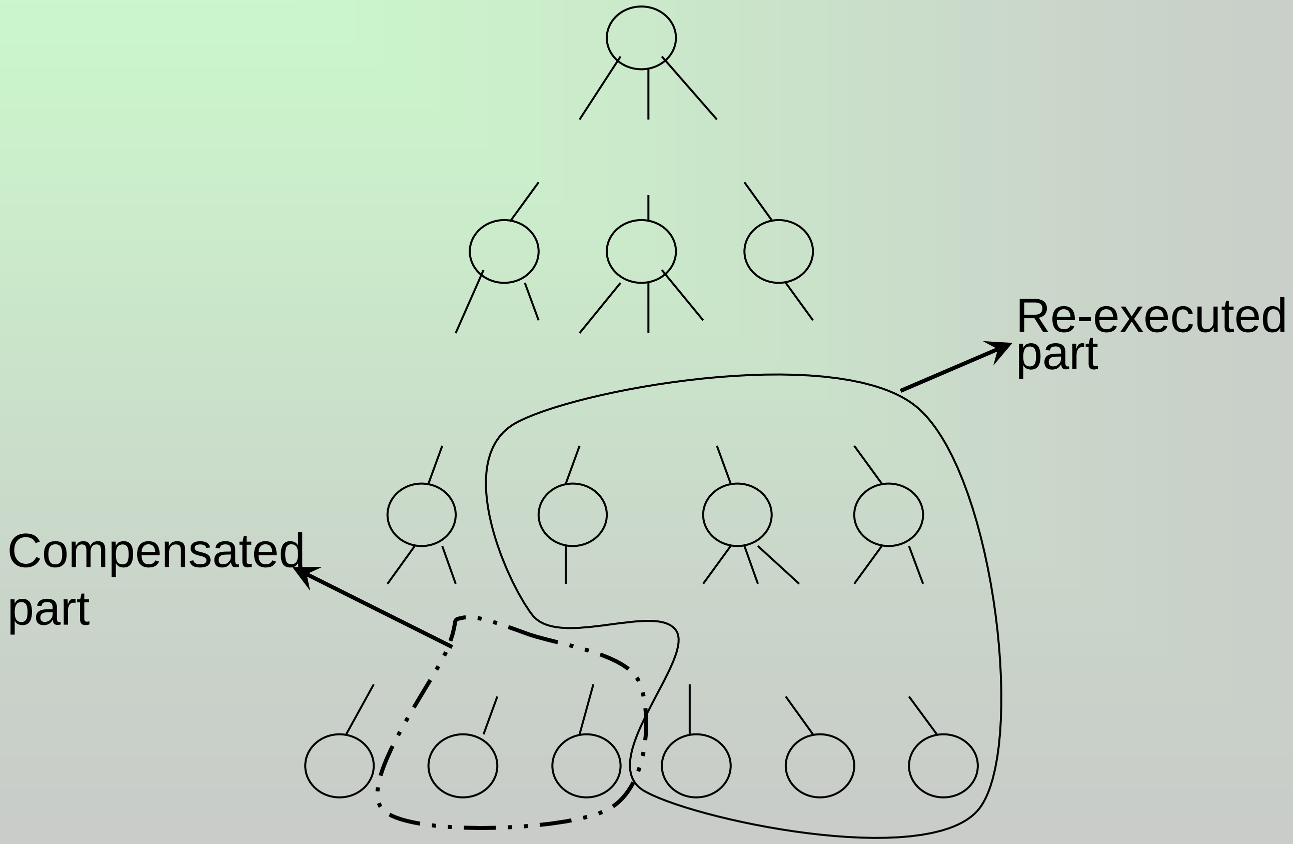
Example – cont’d

- House-Building Contract: Weak & Strong Commit Points
  - The re-execution and compensation details are given in the table. Re-executions carried out after weak commits are stated as retries. Weak and strong commit points for the various activities are given as below
    - On weak commit of activities 1 and 2, the acquired materials cannot be returned (activities cannot be compensated).
    - When the wall is constructed according to satisfaction, activities 1 to 4 are strongly committed.
    - Note that activities 3 and 4 are directly strongly committed, without earlier weak commit.
    - For the paint job, weak commit occurs when it is found that another undercoat is not required, and strong commit occurs when the painting is completely satisfactory.

Example – cont'd

- House-Building Contract: Payments
  - The table states
    - the terms for three payments
    - when the three payments are due
    - the requirement that the first two payments must be received before painting of the wall can start
  - The costs for the execution, including re-executions and compensations, are included in the payment descriptions

Partial backward recovery – General case



Dependencies

- The transactional properties (defined in the composition model) enable identifying the dependencies that arise between the executions of the activities in a precise and elaborate manner.
- The dependencies deeply impact both the recovery and commitment aspects.
- (This study will be helpful in monitoring behavioral conditions stated in e-contracts during execution.)

Dependencies between executions

- Factors contributing to transactional properties in an execution of each activity are:
    - (Changes in) st-predicate and ce-predicate;
    - (Different) s-terminations and (different) f-terminations;
    - Beginning of execution;
    - Weak commit and strong commit; and
    - Compensation and re-execution.
- Dependencies involving each of these factors in executions of activities can be defined. Most combinations are possible
- Here, we explain some dependencies with an example.

Procurement example

- This concerns with procurement of a set of windows for a house under construction.
- The order will contain a detailed list of the number of windows, the size and type of each of them and delivery date.
- The type description may consist of whether part of the window can be opened and, if so, how it can be opened, insulation and draft protection details, whether made up of single glass or double glass, etc.
- The activities are described in the following. The execution-tree is simply a path containing nodes for each of the activities in the given order.

Procurement activities

- P1. Buyer: Order Preparation – Prepare an order and send it to a seller.
- P2. Seller: Order Acceptance – Check the availability of raw materials and the feasibility of meeting the due date; if both are satisfactory, then accept the order.
- P3. Seller: Arrange Manufacturing – Forward the order to a manufacturing plant.
- P4. Plant: Manufacturing – Manufacture the goods in the order.
- P5. Plant: Arrange Shipping – Choose a shipping agent (SA) for shipment of the goods to the buyer.
- P6. SA: Shipping - Pack and ship goods.
- P7. Buyer: Check Goods – Verify that the goods satisfy the prescribed requirements.
- P8. Buyer: Make Payment – Pay the seller.

Dependencies-1

- P1. Buyer: Order Preparation
- P2. Seller: Order Acceptance – if raw materials available and due date feasible
- P3. Seller: Arrange Manufacturing – Forward to a manufacturing plant.
- P4. Plant: Manufacturing – Manufacture the goods in the order.
- P5. Plant: Arrange Shipping – Choose a shipping agent (SA).
- P6. SA: Shipping - Pack and ship.
- P7. Buyer: Check Goods – Verify goods satisfy the requirements.
- P8. Buyer: Make Payment – Pay.

- Once the order is accepted: If it cannot be cancelled, but can be modified (delivery date/quantity changed), then on s-termination of P2, weak-commit P1 and P2.
- There may also be a dependency: the execution of P3 can begin only on weak-commitment of P2.
- If order cancellation is possible, postpone weak commitment of P1 and P2.
- In the following, we assume that the order cannot be cancelled.

Dependencies-2

- P1. Buyer: Order Preparation
- P2. Seller: Order Acceptance – if raw materials available and due date feasible
- P3. Seller: Arrange Manufacturing – Forward to a manufacturing plant.
- P4. Plant: Manufacturing – Manufacture the goods in the order.
- P5. Plant: Arrange Shipping – Choose a shipping agent (SA).
- P6. SA: Shipping - Pack and ship.
- P7. Buyer: Check Goods – Verify goods satisfy the requirements.
- P8. Buyer: Make Payment – Pay.

- The plant may find that the goods cannot be manufactured according to the specifications, i.e., P4 fails.
- If the failure is due to inability to produce the required quantity by the due date, then the buyer may be requested to postpone the due date or reduce the quantity or both (change in st-predicate of P1).
- (Similar situation arises when the buyer wants to update the order by increasing the quantity.)
- This will result in a re-execution of P1 followed by a re-execution of P2. Then the past execution of P4 may be successful or a re-execution may be done. Weak commitments of P1 and P2 allow for such adjustments.

Dependencies-3

- P1. Buyer: Order Preparation
- P2. Seller: Order Acceptance – if raw materials available and due date feasible
- P3. Seller: Arrange Manufacturing – Forward to a manufacturing plant.
- P4. Plant: Manufacturing – Manufacture the goods in the order.
- P5. Plant: Arrange Shipping – Choose a shipping agent (SA).
- P6. SA: Shipping - Pack and ship.
- P7. Buyer: Check Goods – Verify goods satisfy the requirements.
- P8. Buyer: Make Payment – Pay.

- If the Buyer finds the goods do not meet the type specifications (or the plant "recalls" due to some defects), that is, P7 fails, then, P4 has to be re-executed. In addition, P5 and P6 have to be re-executed: the buyer ships back old goods to the plant and the plant ships new goods to the buyer.
- An example is: two of the windows have broken glasses and a wrong knob was sent for a third window. (The knob has to be fixed after mounting the window.) Then, replacements for the two windows have to be made (in P4), the damaged windows and the wrong knob have to be picked up and the new ones delivered: if by the same shipping agent, the re-execution of P5 is trivial.

Dependencies-4

- P1. Buyer: Order Preparation
- P2. Seller: Order Acceptance – if raw materials available and due date feasible
- P3. Seller: Arrange Manufacturing – Forward to a manufacturing plant.
- P4. Plant: Manufacturing – Manufacture the goods in the order.
- P5. Plant: Arrange Shipping – Choose a shipping agent (SA).
- P6. SA: Shipping - Pack and ship.
- P7. Buyer: Check Goods – Verify goods satisfy the requirements.
- P8. Buyer: Make Payment – Pay.

- The shipping agent is unable to pack and ship goods at the designated time, that is, P6 fails. Then either the delivery date is postponed (adjustment in the st-predicate of P1) or the plant may find another shipping agent, that is, P5 is (compensated and) re-executed. In the latter case, it follows that P6 will also be (compensated and) re-executed

Dependencies Summary - 1

I. Any of the compensation, weak commit and strong commit actions on one activity may require any of these three actions for another activity.

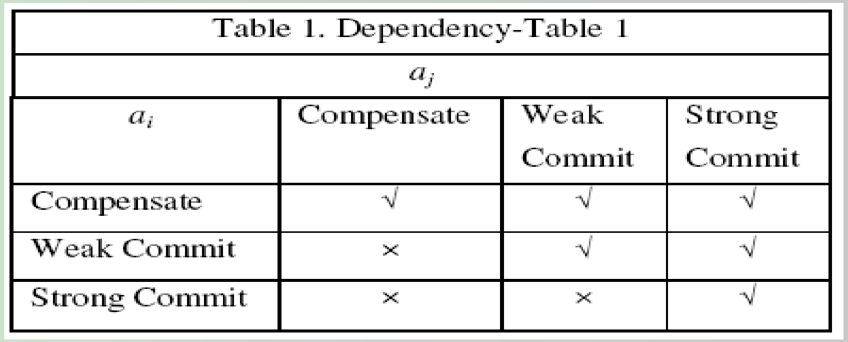


Table 1. Dependency-Table 1

a_i	a_j		
	Compensate	Weak Commit	Strong Commit
Compensate	√	√	√
Weak Commit	×	√	√
Strong Commit	×	×	√

II. Several dependencies which involve re-execution are also possible. We arrive at a general form in several steps.

- An f-termination of an activity changes the st-predicate of another activity and, in fact, of several activities.
- Each different type of f-termination of an activity changes the st-predicates of a set of activities in a specific way.
- A specific (s- or f-) termination of an execution changes the st-predicates of a set of activities in a specific way.

Dependencies Summary - 2

III. We can also state dependencies of the following type.

- A specific (s- or f-) termination of an activity triggers compensation, weak commit or strong commit of executions of some other activities.
- The (compensate, re-execute, weak commit and strong commit) actions on *ai* change the st-predicates of some other activities.

IV. Dependencies constraining the beginning of an execution of an activity can also be defined.

- For example, for activities *aj* and descendent *ai* possible dependencies are: *ai* cannot begin execution until *aj* (i) s-terminates, (ii) weak-commits, or (iii) strong-commits.

Multi-level model - Composition

- Composition **C** is a tree.
- Nodes in the tree are (sub-)compositions of basic or composite activities; Compositions of composite activities are, again, trees. Thus **C** is a "nested" tree.
- An st-predicate is associated with **C**. From this, st- and ce-predicates of all the nodes of **C** are derived.

Multi-level Model - Composite Activity

- Execution of each sub-composition of **C** yields an execution-tree, called composite activity tree (c-tree). To put these trees together, each c-tree is converted to a one source one sink acyclic graph by adding edges from the leaves of the tree to a single (dummy) sink node. We call this a closed c-tree.
- Execution of **C** yields a closed c-tree whose nodes correspond to executions of activities (which themselves are closed c-trees). Thus, the graph can be expanded until all the nodes correspond to basic activities.

Multi-level Model – Transactional Properties

- At each individual level, for each node, the transactional properties are applicable. After the recovery of one node, the recovery efforts at the parent level execution will continue.
- Compensation of a composite activity may involve execution of a composition that does the compensation. This is also specified as a tree with suitable st-predicate.
- Retrying a composite activity may involve a partial backward recovery followed by a forward recovery. The forward recovery may require adding additional sub-trees at some nodes and specifying the st- and ce-predicates for the nodes in them, and adjusting the ce-predicates of other nodes appropriately.

Transactional properties (Cont’d)

- An execution a-i is
  - (locally) compensatable if the execution can be undone to get the null termination.
  - compensatable relative to **C** if either it is locally compensatable or it can be compensated by executing a compensating activity within **C**.
  - (locally) retriable if there is a re-execution that will yield an s-termination.
  - retriable relative to **C** if it is locally retriable or additional activities can be executed in **C** to get the effects of an s-termination of a-i.
- Weak and strong commitments and atomicity are also defined both locally and relative to **C**.

Example of "relative to" aspect in compensatability

- Let *U* be a composite activity consisting of
  - (i) writing and printing a letter,
  - (ii) preparing an envelope – composite activity *C* made up of
    - (a-1) printing From and To addresses on an envelope with a printer,
    - (a-2) affixing a stamp on the envelope,
  - (iii) inserting the letter in the envelope and sealing it.
- The activity a-2 of affixing a stamp is not compensatable relative to *C*, if the stamp cannot be removed.
- However, *C* may be compensatable relative to *U*, amounting to tearing up the envelope and bearing the loss of the stamp. In that case, we also say that a-2 is compensatable relative to *U*.

Dependencies between activities at different levels

- Compensatability, retriability and weak and strong commitments of an execution of an activity can be defined relative to different ancestors of that activity.
- These (extended) definitions of the transactional properties can be used to define dependencies between activities at different levels of the composition.

Multi-level commitment and closure

- Compensatability, retriability and weak and strong commitments of C are all relative to U.
- The execution stages in the diagram, given for basic activities, are applicable to composite activities also.
- Closure of an activity is independent of the closure of its parent or children activities.
  - A contract for building a house may be closed after the warranty period during which the builder is responsible for repairs.
  - A sub-contract for maintaining an air-conditioning system in that house may close at a different time.

st- and ce-predicates

- These are activity-dependent.
- We can expect that they can be expressed more precisely for some activities than for some others.
- For some activities, what constitutes an s-termination may not be known until after the execution of that activity, and even after the execution of many subsequent activities.
- Syntactic specification of ce-predicate may be made more precise, with an appropriate language (which would have constructs for specifying Boolean connectives and priorities).

Capabilities for execution adjustment

- In a multi-level set up, the activities that are re-executed or rolled back would, in general, be composite activities, that too executed by different parties autonomously.
- Therefore, the choices for re-execution and roll back may be limited, and considerable pre-planning may be required in the design phase of the contract.

Multi-level Model Discussion - 1

- Our primary goal is embedding transactional properties in executions of e-contract activities.
- Dealing with (hierarchically) composite activities is inevitable.
- Dependencies between executions of activities in the same level or different levels need to be complied with during execution of the contract.
- This work identifies several dependencies in a systematic manner using a multi-level composition model.

Multi-level Model Discussion - 2

- Level-wise definitions of compensatability and retriability clarify the properties and requirements in the executions of activities and sub-activities, in contracts and sub-contracts.
- This helps in delegating responsibilities for satisfying the required properties in the executions to relevant parties precisely and unambiguously.

Multi-level Model Discussion - 3

- The transactional properties in our model can be used to refine the conditions for the closure of the contract.
- Features such as "the life of a contract may extend far beyond the termination of the execution of the activities in the contract" can be accommodated fairly easily in our model.
- Terms of payments for the activities can be related to the execution states of the activities.
- We believe that our transactional properties will be useful in other applications also, with electronic and/or non-electronic activities.

Summary

- Several proposals exist in the literature for
  - Atomic execution of a group of activities,
  - Using application semantics to determine whether the result of execution (success of some activities and failure of the others) is a successful termination of the whole group or not,
  - Doing compensation at different levels, etc.
- We bring the semantics in terms of guaranteed termination and atomicity by
  - using backward-recovery (compensation) and forward-recovery (retriability)
  - at the various hierarchical levels of the composition.

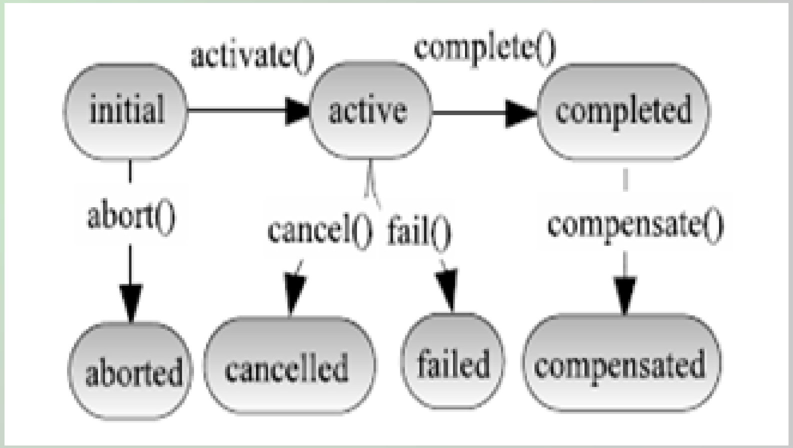
Recent Transaction Model Proposals for Services

Web Services Composition with Transactional Requirements

- Proposed by:
  - [S. Bhiri, O. Perrin, and C. Godart, 2005]
- And further work done by:
  - [Frederic Montagut and Refik Molva, Augmenting Web Services Composition with Transactional Requirements, 2006].

Transactional Web Services - 1

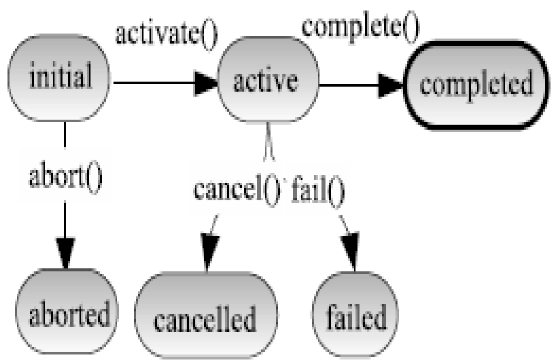
- Four properties of services are defined:
  - Retriable (r), Compensatable (c), Retriable and Compensatable (rc) and Pivot (p).
- A service can combine properties. The combinations are:
  - {r; cp; p; (r, cp); (r, p)}.
- A state/transition model is used for the internal behavior of a service.
  - States are: {initial, active, aborted, cancelled, failed, completed, compensated}.



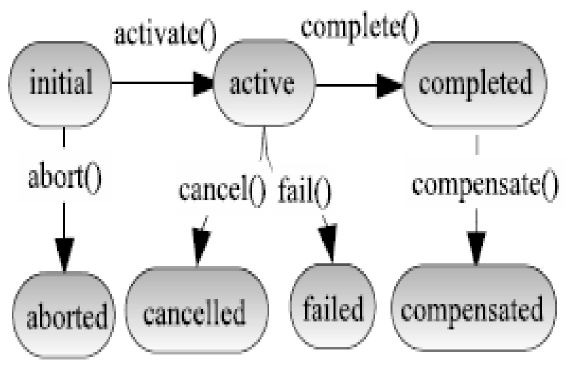
Transactional Web Services - 2

- External transitions are: {activate(), abort(), cancel(), compensate()}.
  - External transitions enable a service to interact with outside and are fired by external entities.
- Internal transitions, fired by the service itself, are: {complete(), fail(), retry()}.
- The considered termination states (ts) are:
  - Failed, completed, compensated, aborted and canceled.
- Transactional properties of services are differentiated by termination states:
  - Failed is not a ts of s iff s is retriable;
  - Compensated is a ts of s iff s is compensatable.

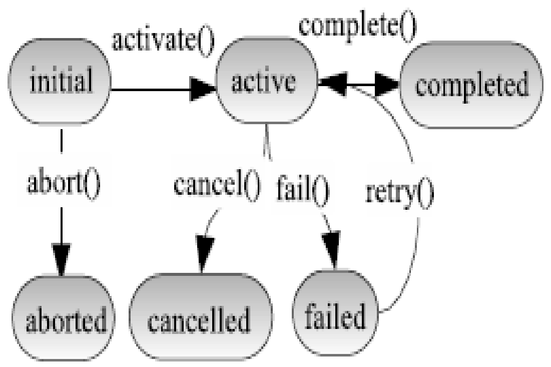
Services States/Transitions



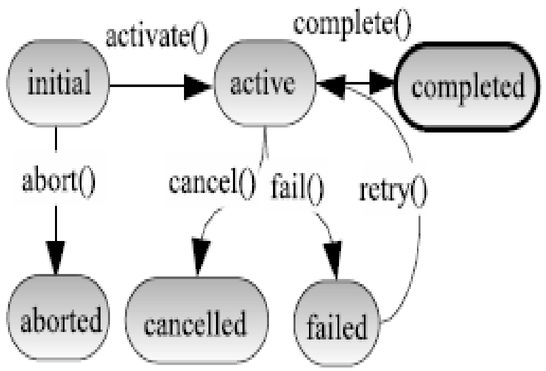
(a) a pivot service



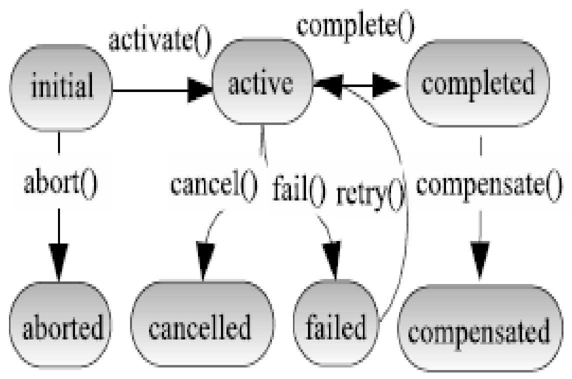
(b) a compensatable service



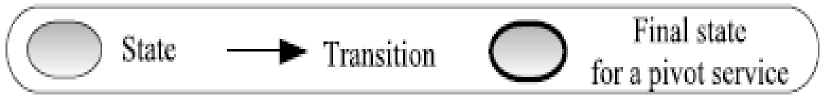
(c) a retriable service



(d) a retriable and pivot service



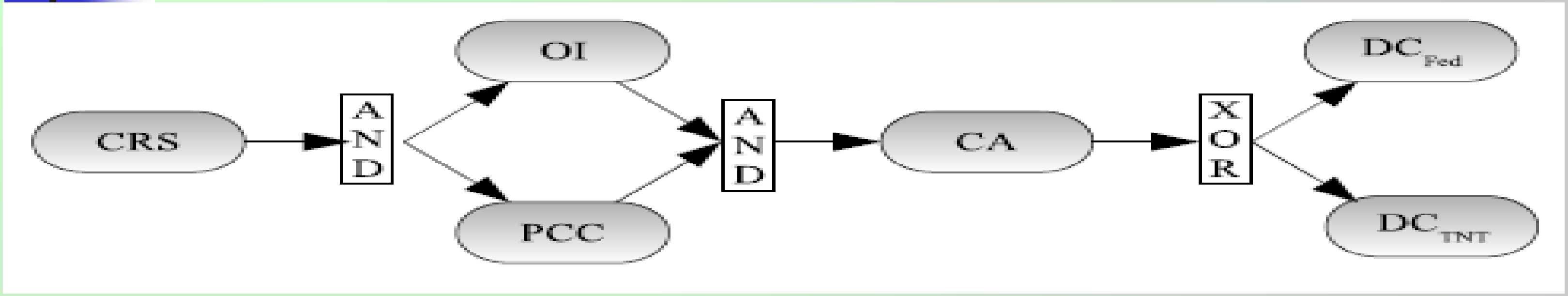
(e) a retriable and compensatable service



Transactional Composite (Web) Service

- Existing Web services are combined to form a composite Web service.
  - "A Transactional Composite (Web) Service (TCS) emphasizes transactional properties for composition and synchronization of component Web services.
  - It takes advantage of services transactional properties to specify mechanisms for failure handling and recovery."
- An Acceptable Termination State (ATS) of a TCS is a set of termination states of the component Web services that are acceptable to the user.

An Example - A composite service for online computer purchase.



- Services involved are:
  - the Customer Requirements Specification (CRS) service used to receive the customer order and to review the customer requirements,
  - the Order Items (OI) service used to order the computer components if the online store does not have all of it,
  - the Payment by Credit Card (PCC) service used to guarantee the payment by credit card,
  - the Computer Assembly (CA) service used to ensure the computer assembly once the payment is done and the required components are available, and
  - the Deliver Computer (DC) service used to deliver the computer to the customer (provided either by Fedex (DCFed) or TNT (DCTNT)).

[Source: Bhiri et al, 2005]

ATS used in the example

services / ats	CRS	OI	PCC	CA	DC_{Fed}	DC_{TNT}
ats_1	(completed,	completed,	completed,	completed,	initial,	completed)
ats_2	(completed,	completed,	completed,	completed,	completed,	initial)
ats_3	(completed,	compensated,	failed,	aborted,	aborted,	aborted)
ats_4	(completed,	failed,	compensated,	aborted,	aborted,	aborted)
ats_5	(completed,	cancelled,	failed,	aborted,	aborted,	aborted)
ats_6	(completed,	completed,	completed,	completed,	failed,	completed)

[Source: Bhiri et al, 2005]

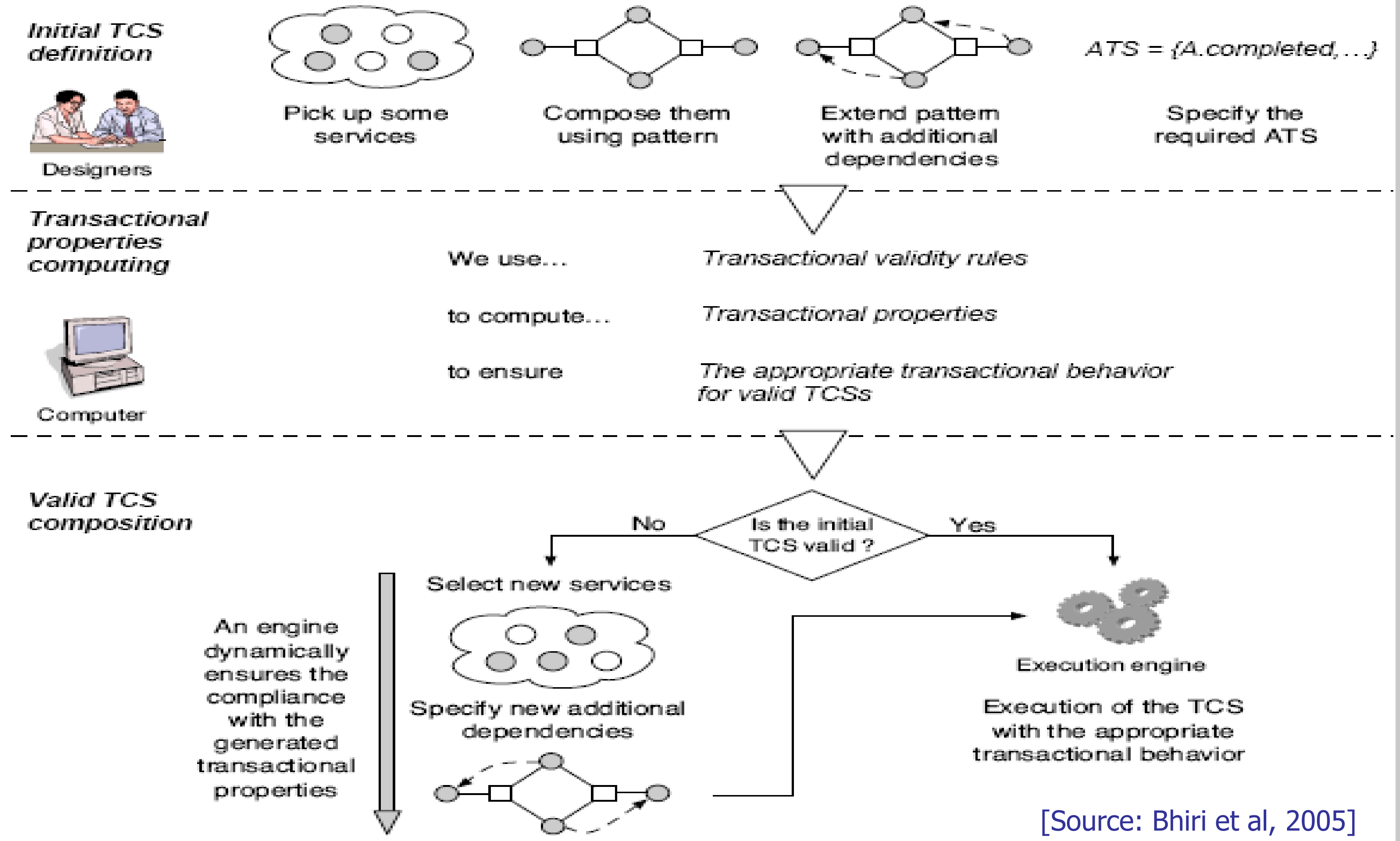
Dependencies between services

- Dependencies are defined between service executions.
- A dependency from s1 to s2 exists if a transition of s1 can fire an external transition of s2.
- The following dependencies have been defined:
- Activation dependency:
  - the completion of s1 => the activation of s2;
- Alternative dependency:
  - the failure of s1 => the activation of s2;
- Abortion dependency:
  - The failure, cancellation or the abortion of s1 => the abortion of s2;
- Cancellation dependency:
  - The failure of s1 => the cancellation of s2.
- The last three are called transactional dependencies.

Objective and Overview

- First, an abstract representation of the composition with desired transactional properties of its constituent services is formulated.
  - This is done by using a set of interactions patterns (sequence, AND-split, AND-join, …)., and specifying the required ATS.
- Appropriate transactional behavior from the TCS skeleton and the ATS is obtained.
  - This is equivalent to identifying the appropriate dependencies between services.
- Then, by a match-making process, concrete Web services satisfying the transactional properties are selected to obtain a TCS.
- The validity of the TCS is checked by “transactional validity rules”.
- If not valid, new TCS is tried.

Objective and Overview - 2


Initial TCS definition
Designers
Pick up some services
Compose them using pattern
Extend pattern with additional dependencies
ATS = {A.completed, . . .}
Specify the required ATS
Transactional properties computing
Computer
We use...
Transactional validity rules
to compute...
Transactional properties
to ensure
The appropriate transactional behavior for valid TCSs
Valid TCS composition
No
Is the initial TCS valid ?
Yes
Select new services
An engine dynamically ensures the compliance with the generated transactional properties
Specify new additional dependencies
Execution engine
Execution of the TCS with the appropriate transactional behavior

[Source: Bhiri et al, 2005]

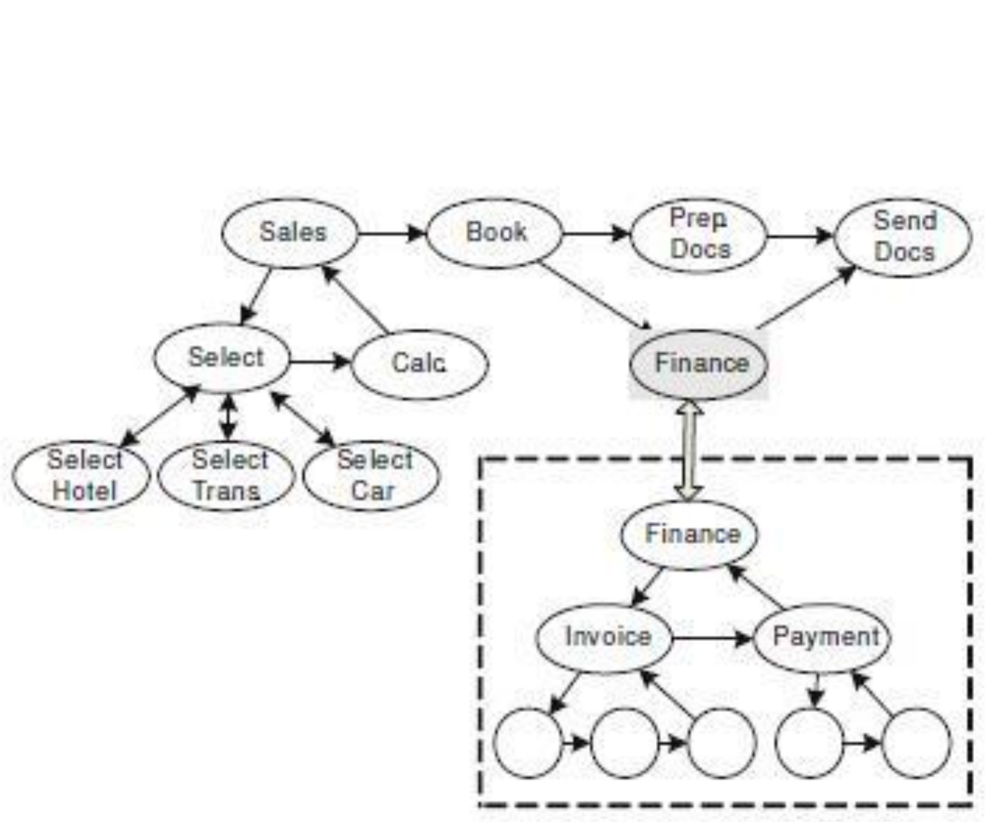
Abstract Transactional Constructs

- Proposed by:
  - Ting Wang, Paul Grefen and Jochem Vonk, Abstract Transactional Construct: Building a Transaction Framework for Contract-Driven, Service-Oriented Business Processes, ICSOC 2006, LNCS Vol. 4294, Springer, pp. 434-439, 2006.
- ATCs are abstract types of existing transaction models that can be composed and executed in a service-oriented transaction framework.
- Business Transaction Framework is built on ATCs.
- By selecting and composing ATCs on demand, flexible and reliable process execution is guaranteed.

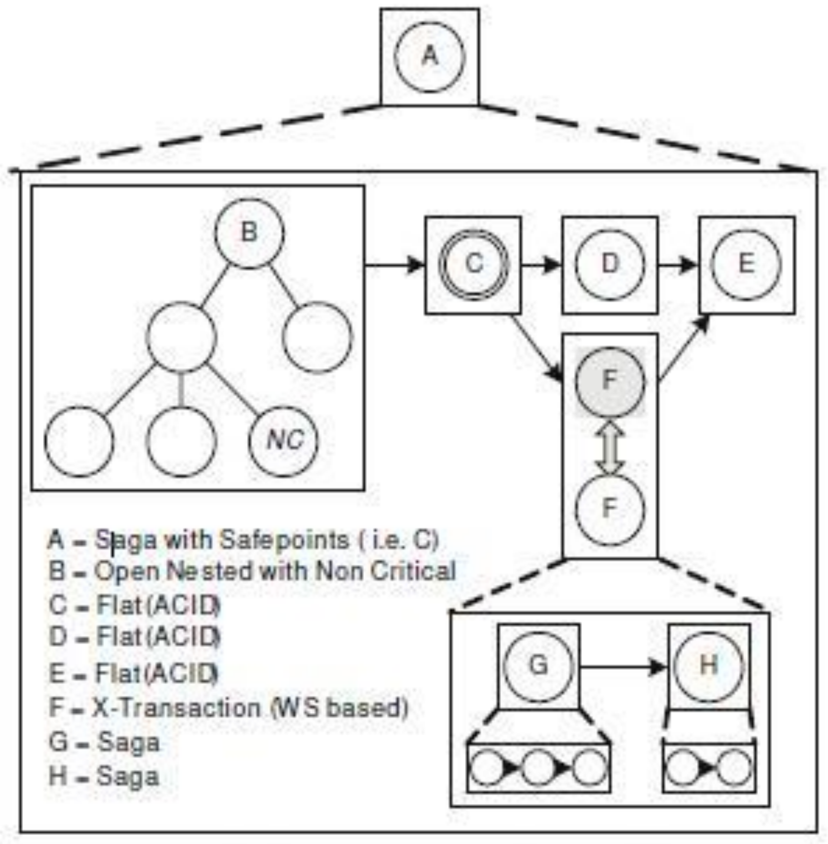
Characteristics of ATCs

- An ATC has an internal structure.
  - Flat – Basic unit with ACID properties
  - Sequence – Chained or Saga transaction
  - Complex – Mixed type of arbitrary sequences and parallels corresponding, roughly, to complex workflow models.
  - Tree – Nested structure with parent-child relationships with no control flow between the nodes.
- ATCs are composed in recursive manner.
- Each ATC guarantees specific transactional qualities
  - Called Transactional Quality of Service (TxQoS).

Example Travel Agency



1-a: Booking Process



A – Saga with Safepoints (i.e. C)
B – Open Nested with Non Critical
C – Flat (ACID)
D – Flat (ACID)
E – Flat (ACID)
F – X-Transaction (WS based)
G – Saga
H – Saga

1-b: Recursive ATC Composition

Source: [ATC-ICSOC-2006]

Example Travel Agency

- Customers create a trip by selecting a hotel, transportation, and an optional car rental (in parallel).
- The costs are calculated and the trip booked.
- In parallel, the required documents are prepared and the financial details (invoicing and payment checking) are dealt with.
- Being a small travel bureau, the financial dealings are outsourced through a Web service.
- ATC composition:
  - Saga-like model comprising 'sales', 'book', 'prep. docs', 'finance', and 'send docs';
  - Some variant of open-nested transaction model for the selection activities;
  - Saga for the internals ('invoice' and 'payment') of the Web service;
  - Etc.

ATC Management

- ATCs are created in the design phase
- ATC templates are stored in ATC library.
- Newly created (composite) ATCs can also be stored for later use and for higher level composition.

Achieving Atomicity Using Commutativity

[Michael Melliar-Smith and Louise E. Moser, 2007]

Motivation - 1

- "Business activities incur the risk of long delays and locked data when using the distributed transaction strategy based on two-phase commit and conservative locking."
  - [Due to blocking nature of 2PC], "if a transaction in one enterprise locks data in the database of another enterprise and then the server of the first enterprise fails, the data in the second enterprise might remain locked for an indeterminate period of time until the server in the first enterprise is recovered from the fault. The risk of such delays is unacceptable, particularly when the other participants in the business activity are unknown or of uncertain dependability. Consequently, in practice, Web Services Atomic Transactions are not used across a wide-area distributed environment."
  - "The problems of distributed transactions, based on the two-phase commit protocol, can be reduced, but not eliminated, by use of the three-phase commit protocol [Skeen and Stonebraker, 1983]. However, the three-phase commit protocol increases transaction processing overhead and latency in the normal fault-free case. Consequently, the three-phase commit protocol is not used in practice."

Motivation - 2

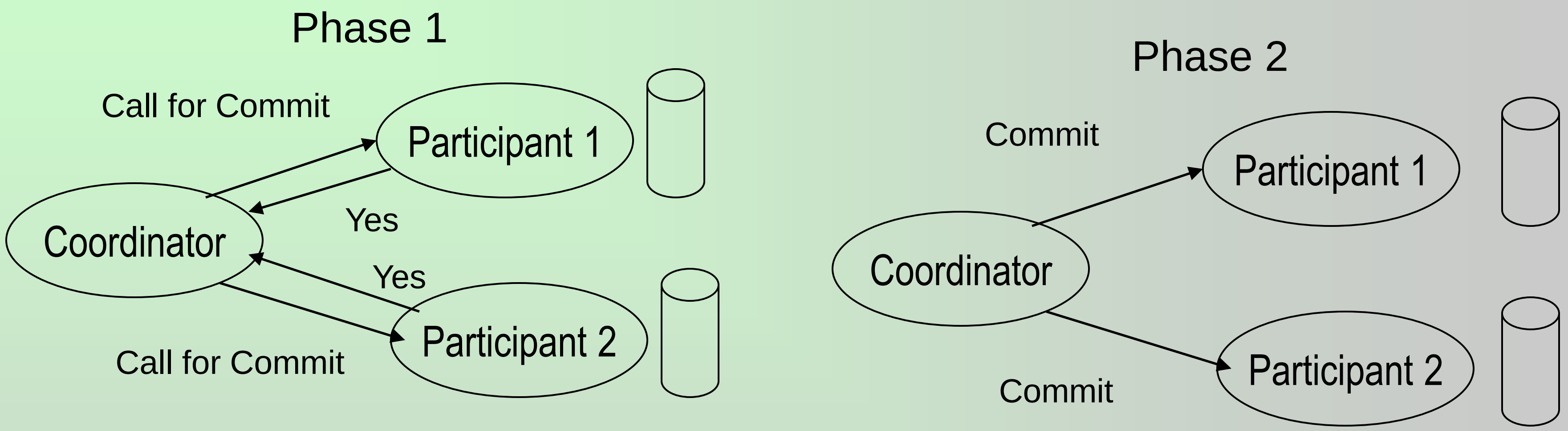
- "The Web Services Business Activity Specification [Cabrera et al. 2005] addresses these problems by means of an extended transactions strategy with compensating transactions [Garcia-Molina and Salem 1987].
- Compensating transactions are difficult to design and program, have a higher error rate, and incur a high risk of leaving the databases in an inconsistent state. Detecting and removing these inconsistencies are difficult, labor-intensive and time-consuming."

Proposed Method

- Their idea is to allow interleaving of only commutative steps.
    - A mechanism to reserve the necessary resources at the beginning of the execution is followed.
    - If all the necessary resources are not available, then the transaction execution will not start.
    - This is akin to getting all the locks at the beginning of the execution.
    - Reservation requests of other transactions will be entertained only if non-conflicting resources are available.
    - Reservations will be held until the transaction commits or aborts.

Protocols for commitment

Two-Phase Commit (2PC)



Phase 1 – Preparation: Coordinator sends a request for commit to all participants and waits until it recieves response from them.

Phase 2 – Commit/Abort: Coordinator decides to commit the transaction, if it receives YES from all participants; and decides to abort, if it receives NO from any of the participants.

2PC Gurantees atomicity in a distributed environment.
2PC involves message communications for Request for vote, voting and decision, so the delay is large.

Limitations and variations of 2PC

- 2PC is a blocking protocol.
  - All participants who voted YES block if the coordinator fails before sending the decision in phase 2.
- Variations of 2PC
  - Presumed Commit and Presumed Abort
    - Designed to reduce the number of messages
    - Assume “default” decisions
    - Differ with respect to logging and recovery details
  - 3PC
    - Non-blocking protocol
    - Increased number of messages
  - Volatile 2PC
    - In this context, 2PC is called Durable 2PC

Limitations and variations of 2PC- 2

- Volatile 2PC (4PC)
  - Phase 1 – Prepare phase of Volatile 2PC
    - Before the transaction starts the Durable2PC, all participants registered with the Volatile2PC are informed and can flush cached data. Any failure at this point will cause the transaction to roll back.
  - Phase 2 & Phase 3 –Prepare & Commit/Rollback phases of 2PC
    - The coordinator then conducts the entire Durable2PC protocol.
  - Phase 4 – Commit/Rollback phase of Volatile 2PC
    - Once the transaction has terminated, the second phase of the Volatile 2PC protocol is executed.
    - Any failures at this stage are ignored as the transaction is terminated, and therefore nothing is affected.
  - All participants registered for volatile 2PC must respond to coordinator with vote messages before coordinator sends prepare messages to cohorts registered for (durable) 2PC.
  - Participant registered for volatile 2PC is not guaranteed to receive commit/abort message from coordinator. (Since it does not support durable resources, the message serves no purpose.)
  - Useful to work on cached objects.

Web Services Transaction Management

- Business Transaction Protocol
- Tentative Hold Protocol
- WS-Transaction
  - WS-Atomic Transaction
  - WS-Business Activity
- WS-Coordination
- WS-Scheduler

Business Transaction Protocol (BTP) - 1

- Developed by OASIS [OASIS 2002]
- Manages complex B2B transactions over Internet
- XML based standard interoperation protocol and follows 2 PC commit protocol
- Supports asynchronous communication between loosely-coupled applications
- Atoms Vs Cohesions
  - Atoms are short duration transactions, follows ACID
  - Cohesions are long duration transactions which are combination of several atom transactions, relaxes ACID

Business Transaction Protocol (BTP) - 2

- Cohesions
  - Cohesive transactions relax isolation property by making intermediate results visible.
  - Cohesive transactions may deliver different termination results (commit or rollback) to its participants. Consistency is determined based on the agreement and interaction between the coordinator and initiator.
  - Initiator is allowed to terminate the transaction.
- BTP incorporates business logic into the transaction infrastructure.
  - It adds business logic between the phases in 2 PC.
  - The intermediate results are visible to other transactions and thus isolation is relaxed.

WS-Transaction - 1

- Defines two models for web service transactions: Atomic transactions and Business activity transactions.
- WS-AtomicTransaction [OASIS 2007]
  - Similar to the traditional ACID transactions, intended for short-lived activities
  - Implements transactional atomicity using 2PC, ensures global atomicity
  - Supports Durable2PC and Volatile2PC
  - Works in a trusted domain

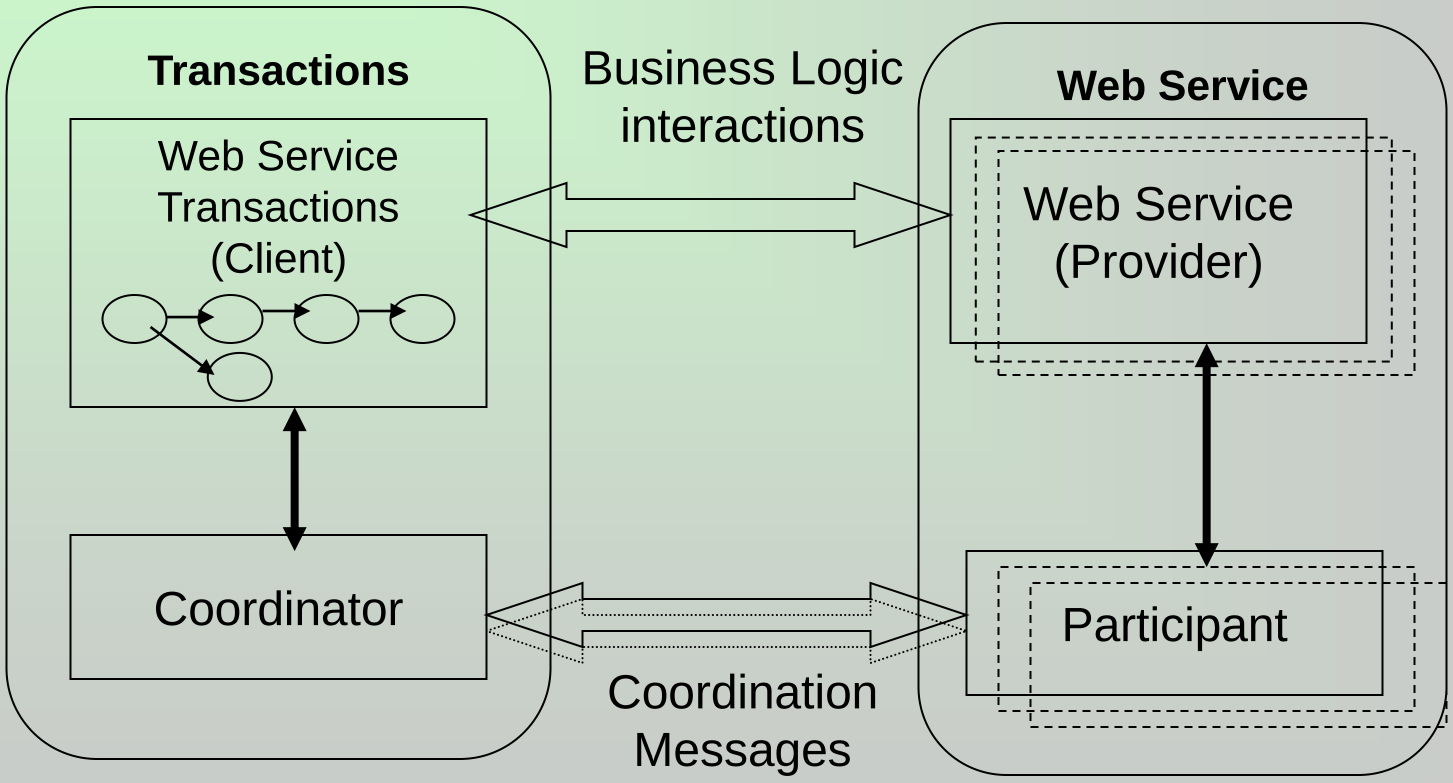
WS-Transaction - 2

- WS-BusinessActivity [OASIS 2007]
  - Based on the Open nested transaction model.
  - Useful when non-atomic outcomes are expected.
  - Ensures consistency through compensation (by parents).
  - Children can proactively communicate with parents without waiting for a request.
  - Intended for loosely-coupled, long-lived activities.
  - Designed for an activity that consists of sequence of tasks, where each task satisfies the constraints of an atomic transaction.
  - Participants might make state transitions durable and visible immediately
    - Compensating actions must be used to reverse actions.
  - Sub-transactions may commit independently
  - In case of sub-transaction failure, concerned participant may decide whether the overall transaction should abort or simply ignore it.

WS-Coordination

- Defines an abstract notion of activities, which are distributed units of work, involving one or more parties (which may be services, components, or even objects).
- Specifies Two components
  - Coordinator
    - Responsible for creating context and coordinating the participants according to the applied protocol.
  - Participant
    - Responsible for communicating with the coordinator according to the applied protocol on behalf of web service.
- It creates a new activity, registers for a service, and selects a protocol (as specified in WS-Atomic Transaction / WS-Business Activity).

WS Transaction



Separates coordination from transaction

WS-Scheduler - 1

- Proposed by Alfari et al, 2009.
- Implements service level concurrency control.
- Scheduler resides on web service provider’s side.
- Detects transactional dependencies
  - Build conflict matrix
  - Handles global dependency cycles

WS-Scheduler - 2

- Responsible for managing concurrent instances of WS-Coordination protocol.
- Consistency of transactions' outcome is ensured using the rules
  - A transaction is only allowed to commit after all its dominant transactions have committed
    - Adds a waiting state in the WS-BusinessActivity specification
  - When a transaction aborts and/or compensates its local activities, the local activities of all its dependent transactions are compensated automatically.

Summary

- These standards are mostly
  - Based on 2 PC protocol and a set of extended transactional models
  - focuses on coordination between Participants
- Parties have to agree to a specific model (BTP (atoms and cohesions), WS-AtomicTransaction and WS-BusinessActivity), etc.) before initiating a service..
- supports exchange of messages according to specified model
- Exploits transactional semantic properties of operations
  - E.g. cancelling an order treated as compensation

Web Services and Business Transactions

- [Michael P. Papazoglou, Web Services and Business Transactions, World Wide Web: Internet and Web Information Systems, Vol. 6, pp. 49-91, 2003].
- Premise: "Process oriented workflow systems and e-business applications require transactional support in order to orchestrate loosely coupled services into cohesive units of work and guarantee consistent and reliable execution."
- Business Transaction Framework: "Motivation is … to orchestrate loosely coupled Web services into a single business transaction by offering transactional support in terms of coordinating distributed autonomous business functionality and guaranteeing coordinated, predictable outcomes for the trading partners participating in a shared business process."

Business processes

- A business process is a set of logically related tasks performed to achieve a well defined business outcome. It "specifies:
  - The potential execution order of operations originating from a collection of Web services;
  - The shared data passed between these services;
  - The trading partners involved in the joint process and their roles with respect to the process;
  - Joint exception handling conditions for the collection of Web services; and
  - Other factors that may influence how Web services or organizations participate in a process."

Business transaction types

- Atomic business transactions (service-atoms)
  - They follow the ACID properties.
  - All participants commit or all abort.
- Long-running business transactions (BTs)
  - These are aggregations of several atomic transactions.
  - They exhibit the characteristics and behavior of open nested transactions.
  - Some participants may commit and some others may abort.
- Different atomicity criteria
  - System-level atomicity
  - Operational-level atomicity
  - Business interaction-level atomicity

System-level atomicity

- Service request atomicity
  - implies that each service provider offers services and operations (and guarantees) that these will complete as an atomic piece of work.

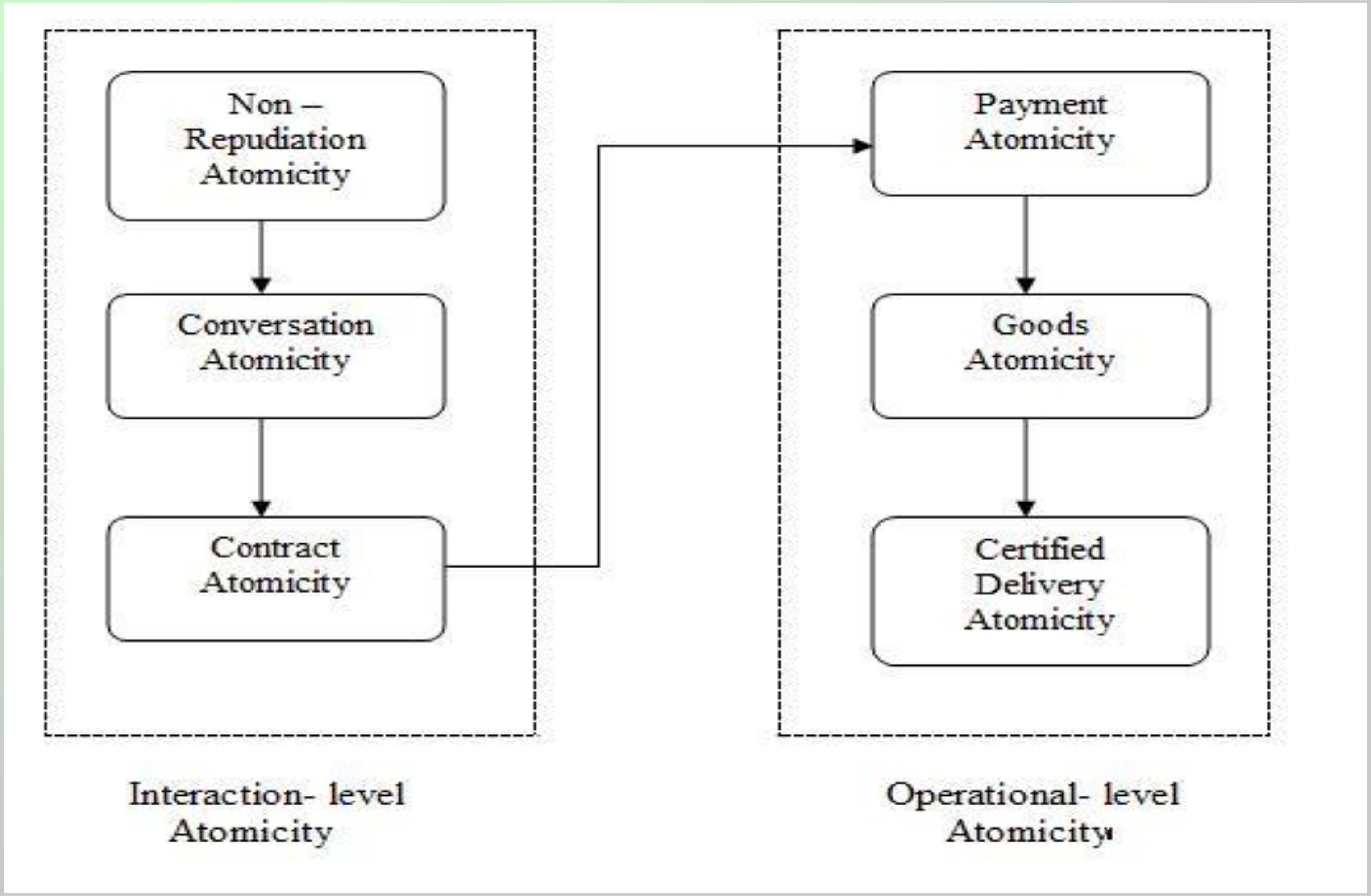
Business interaction-level atomicity

- Non-repudiation atomicity
  - Digitally signing the content of the transaction at an application level
- Conversation atomicity
  - Uses architected conversation messages and headers to begin the conversation and end the conversation.
  - On rollback, each participant (service) will undo the operations it has performed within the conversation and expect its counterpart service to revert back to a consistent state.
- Contract atomicity
  - This is normally based on electronic business protocols that include the exchange of financial information services and the exchange of bills and invoices.
  - If the BT succeeds, it is legally binding.

Operational-level atomicity

- Payment atomicity
  - For many e-business applications, payment-atomic protocols must also be contract-atomic.
- Goods atomicity
  - Payment-atomic and also exact transfer of goods for money.
  - Goods will be received only if payment has been made.
- Certified delivery atomicity
  - Goods-atomicity and the requirement that the right goods are delivered.

Classification of BT Atomicity Types

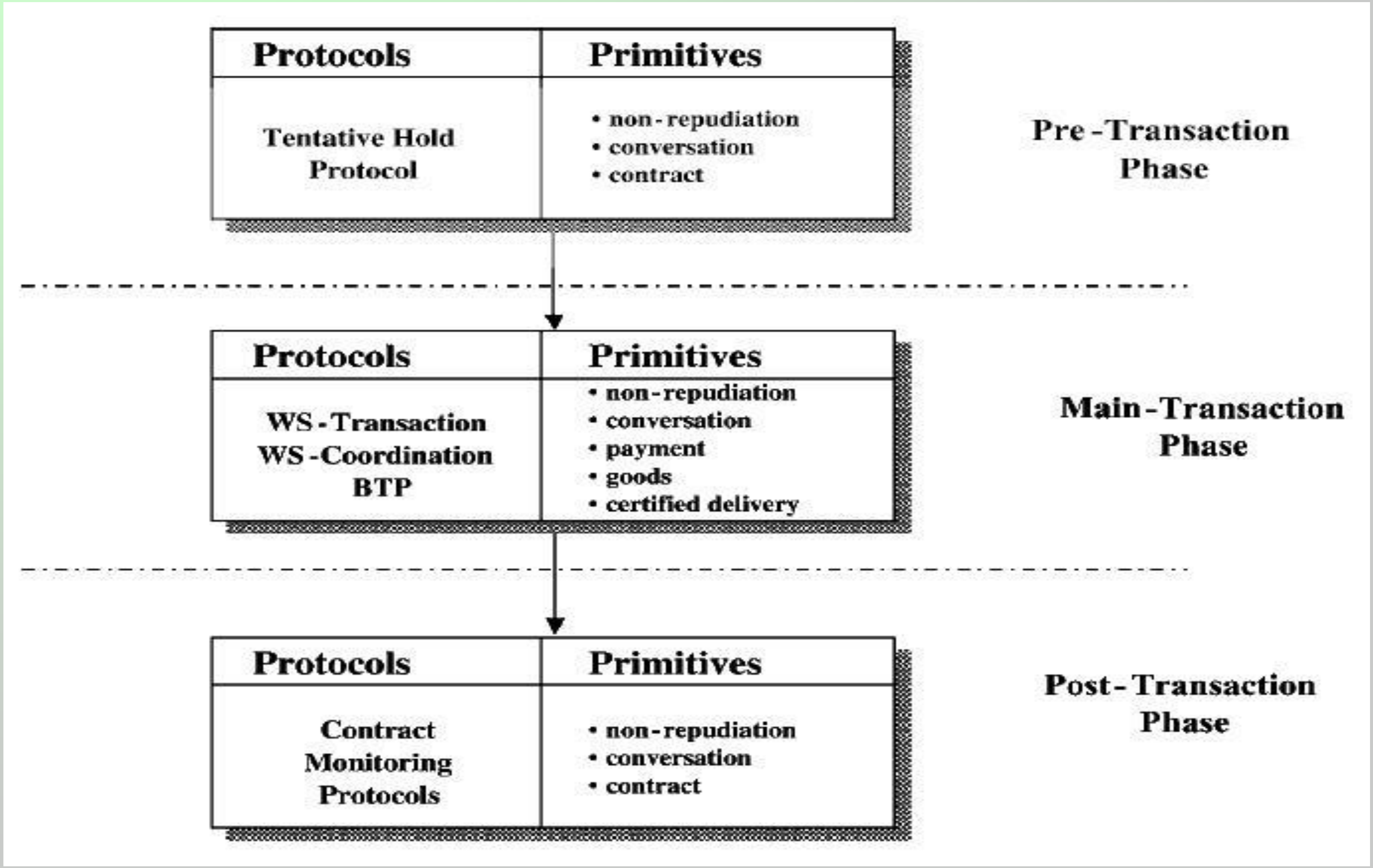

Non – Repudiation Atomicity
Conversation Atomicity
Contract Atomicity
Payment Atomicity
Goods Atomicity
Certified Delivery Atomicity
Interaction- level Atomicity
Operational- level Atomicity

Source: [Papazoglou, 2003]

Business transaction phases

- Pre-transaction phase
  - Meaningful business terms such as order information, prices, delivery conditions and so on are exchanged between trading partners and accepted.
- Main-transaction phase
  - Provides the protocols and infrastructure that coordinates the execution of distributed operations in a Web services environment.
- Post-transaction phase
  - Contract fulfillment phase
  - Monitors the performance of the contract.

BT Phases



Source: [Papazoglou, 2003]

Business process orchestration

- Baseline level specifications (SOAP, UDDI and WSDL) provide the infrastructure that supports the publish, find and bind operations in service-oriented architecture and higher level specifications.
- They enable users to connect different services across organizational boundaries in a platform and language independent manner.
- However, they do not allow defining the business process orchestration semantics of Web services. Higher-level specifications are required.
- BPEL has emerged as the standard to define and manage business process activities and business interaction protocols comprising collaborating Web services.

Tentative Hold Protocol

- A non-blocking protocol
- useful to place a hold on a resource by multiple participants and thus eliminate blocking problems.
- Allows tentative, non-blocking reservations
  - Commit of a resource by one participant will be notified immediately to all other participants who placed hold on the same resource.
- Facilitates automatic co-ordination between two or more business transactions.
- Provides open, loosely coupled, messaging-based framework for information exchange between participants prior to the execution of the actual transaction itself.

Cloud Computing

References

- [Wei et al.]: Zhou Wei, Guillaume Pierre and Chi-Hung Chi, Scalable Transactions for Web Applications in the Cloud, http://www.globule.org/publi/STWAC_europar2009.pdf
- [Das et al.]: Sudipto Das, Divyakant Agrawal and Amr El Abbadi, ElasTras: An Elastic Transactional Data Store in the Cloud, http://www.usenix.org/event/hotcloud09/tech/full_papers/das.pdf
- [Aguilera et al.]: Marcos Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch and Christos Karamanolis, Sinfonia: A New Paradigm for Building Scalable Distributed Systems, SOSP '07.

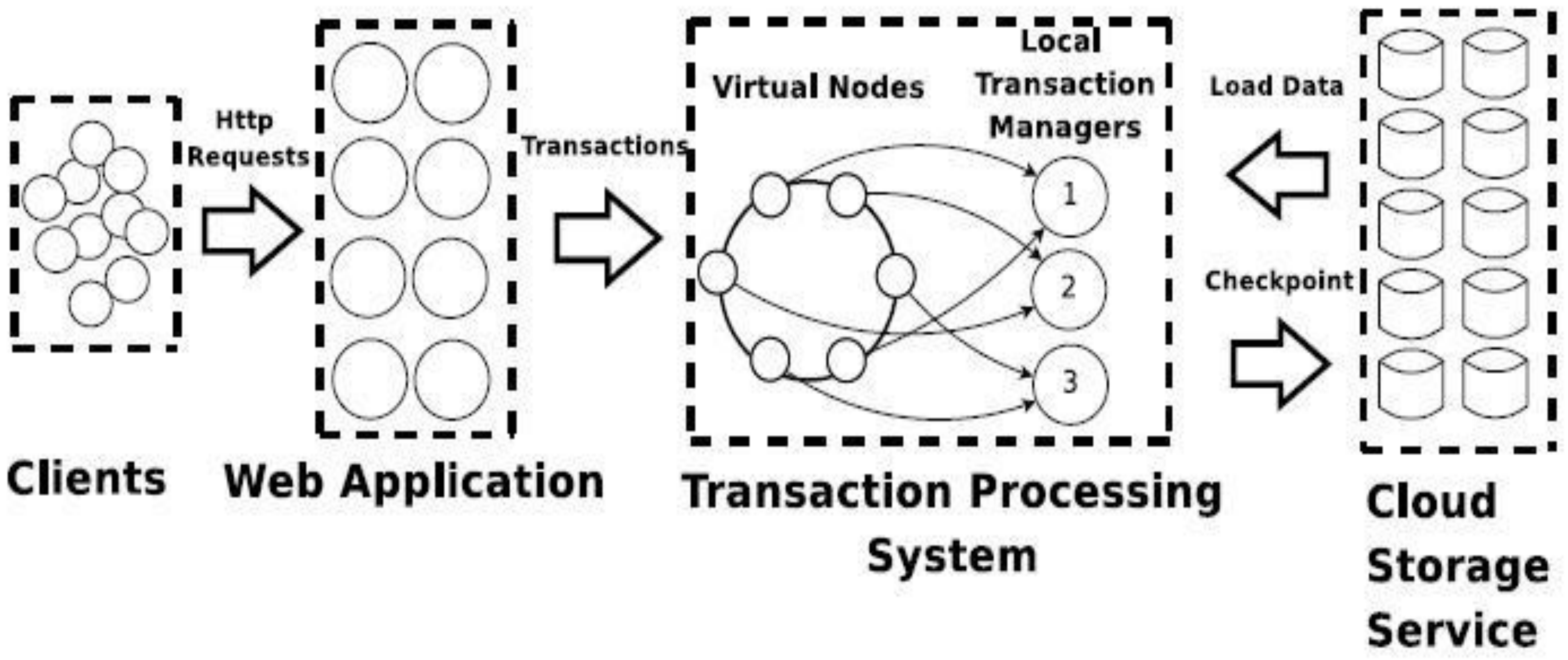
Data Consistency

- Cloud platforms offer scalability and high availability, but only eventual data consistency.
- Several applications such as payment services and online auction services require stronger data consistency.
- [Wei et al.] discuss how to achieve transactional data consistency (with ACID properties).
- They observe that:
  - Transactions in Web applications are short-lived and also access small number of well-identified data items;
  - This implies that two-phase commit protocol can be confined to a relatively small number of servers holding the accessed data items and also that the number of (read/write, write/write) conflicts is low.

Proposal in [Wei et al.]

- Transaction management is provided on secondary copy of the application data, loaded from the cloud storage service.
- The transaction manager is split into several Local Transaction Managers (LTMs).
- To maintain ACID properties even in the case of server failures
  - Data items and transaction states are replicated in multiple LTMs,
  - Updates from transactions are kept in the memory of LTMs, and periodically checkpointed back to the cloud storage service.

System model

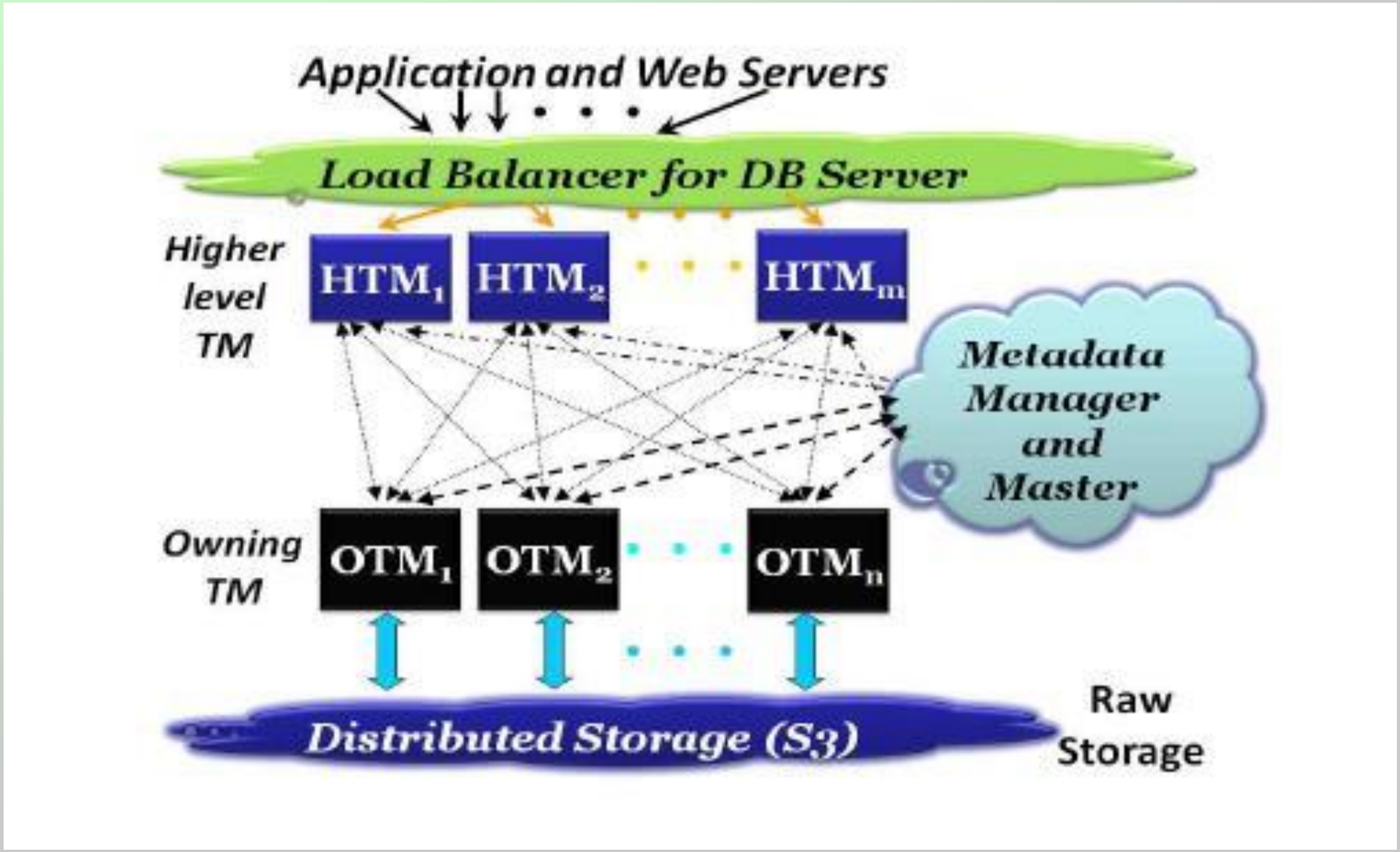


Source: [Zhou Wei, Guillaume Pierre and Chi-Hung Chi]

ElasTraS [Das et al.]

- Focus is on IaaS (Infrastructure as a Service) cloud that provides compute cycles, storage, network bandwidth, etc.
- Addresses scalability and elasticity of the data store while providing transactional data access.
- “The elastic nature of the cloud allows resources to be allocated on demand allowing applications to easily scale up and down with the load changes.”

Overview of the ElasTraS system



Source: [Sudipto Das, Divyakant Agrawal, Amr El Abbadi]

Design overview - 1

- Distributed Storage
  - This takes care of replication and fault-tolerance.
  - The application accessing the storage should ensure that the same object or file is not being written to concurrently.
  - It is assumed that reads and writes to the storage layer are atomic.
- Owning Transaction Manager (OTM)
  - Each OTM owns (really leases) a partition of the data store.
  - It is responsible for all the concurrency control and recovery functionality for its partition.
  - Durability of committed transactions is ensured.

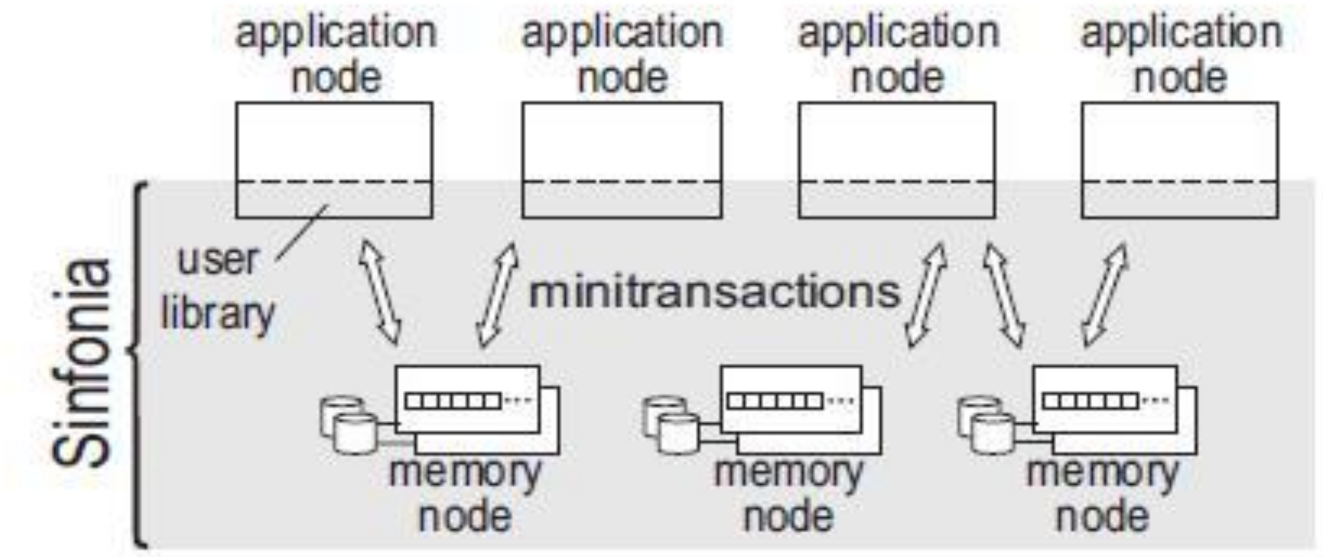
Design overview - 2

- Metadata Manager and Master (MMM)
  - Stores system state, i.e., partition information, mapping of partitions to OTM, leasing information for the OTMs to deal with failures, etc.
  - Provides strong durability and consistency guarantees to the metadata.
  - The Master monitors the system and ensures that if an OTM fails then another OTM is instantiated to serve the partition, and also deals with partition assignment for load balancing.
  - High consistency of the data stored in the MMM is guaranteed through synchronous replication of the contents.
  - High availability can be achieved by judicious placement of replicas so that correlated failures do not affect a majority of the replicas.

Design overview - 3

- Higher level Transaction Managers (HTMs)
  - Execute read-only transactions with a cache of subsets of the database.
  - Coordinate executions of minitransactions (by sending them to appropriate OTMs).
  - Do not have any state coupling with OTMS, and the number of OTM and HTM instances can be different depending on the system configuration and the load on the system.
- Transaction Management
  - Only read-only transactions and minitransactions are supported, the former executed in HTMs and the latter in OTMs.
  - Failures of OTMs and HTMs are handled separately and independently.
- Elasticity
  - Achieved by different number of HTM and OTM instances depending on the load.

Sinfonia [Aguilera et al.]



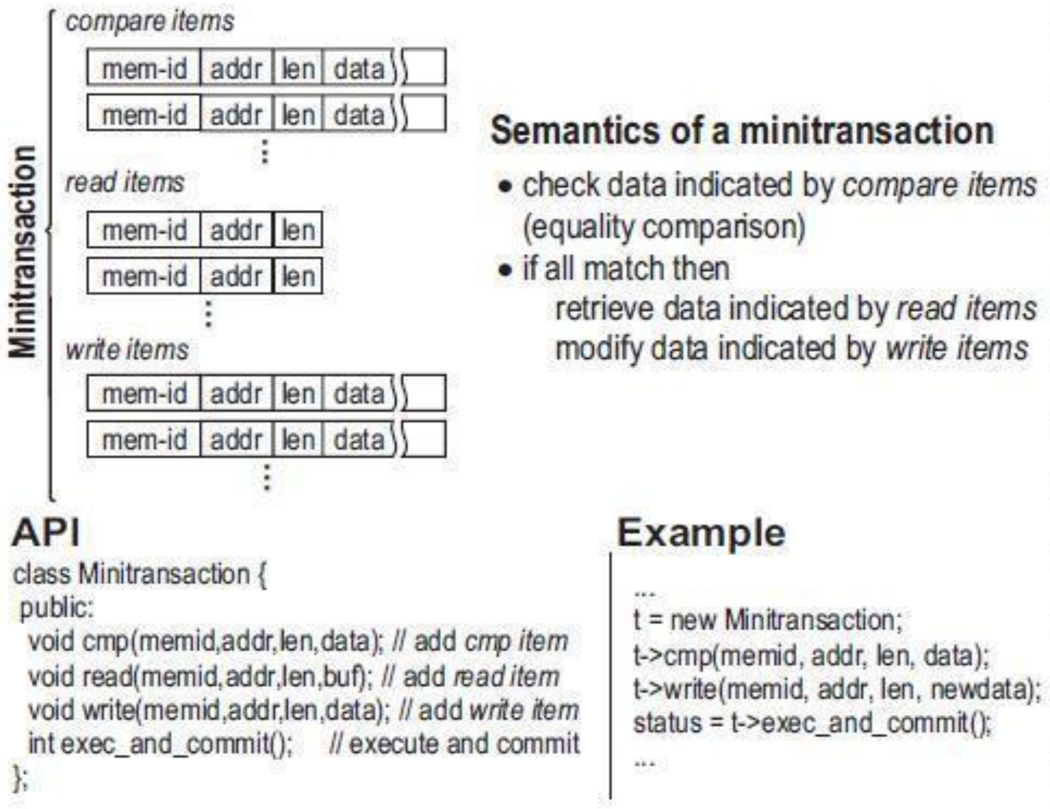
Sinfonia allows application nodes to share data in a fault tolerant, scalable, and consistent manner.

Source: [Marcos K. Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch, Christos Karamanolis]

Basic components

- Sinfonia consists of a set of memory nodes and a user library that runs at application nodes.
- Memory nodes hold application data, either in RAM or on stable storage, according to application needs.
- Application nodes execute a special type of transactions, called minitransactions on memory nodes.
- The user library communicates with memory nodes through remote procedure calls, on top of which the minitransaction protocol is run.
- Both the entire execution and the commitment of a minitransaction are done within a two-phase commit protocol.
- The protocol reflects some different failure assumptions in Sinfonia, requires new schemes for recovery and garbage collection, and incorporates minitransaction execution and a technique to avoid minitransaction deadlocks.

Minitransactions


Minitransaction
compare items
mem-id | addr | len | data
mem-id | addr | len | data
read items
mem-id | addr | len
mem-id | addr | len
write items
mem-id | addr | len | data
mem-id | addr | len | data

Semantics of a minitransaction

- check data indicated by *compare items* (equality comparison)
- if all match then
  - retrieve data indicated by *read items*
  - modify data indicated by *write items*

API

```
class Minitransaction {
 public:
  void cmp(memid,addr,len,data); // add cmp item
  void read(memid,addr,len,buf); // add read item
  void write(memid,addr,len,data); // add write item
  int exec_and_commit();      // execute and commit
};
```

Example

```
...
t = new Minitransaction;
t->cmp(memid, addr, len, data);
t->write(memid, addr, len, newdata);
status = t->exec_and_commit();
...
```

Minitransactions have compare items, read items, and write items. Compare items are locations to compare against given values, while read items are locations to read and write items are locations to update, if all comparisons match. All items are specified before the minitransaction starts executing. The example code creates a minitransaction with one compare and one write item on the same location –a compare-and-swap operation. Methods cmp read, and write populate a minitransaction withour communication with memory nodes until exec_and_commit is invoked.

Source: [Marcos K. Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch, Christos Karamanolis]

Examples of minitransactions

- Swap
  - A read item returns the old value and a write item replaces it.
- Compare-and-swap
  - A compare item compares the current value against a constant; if equal, a write item replaces it.
- Atomic read of many data
  - Done with multiple read items.
- Acquire a lease
  - A compare item checks if a location is set to 0; if so, a write item sets it to the (non-zero) id of the leaseholder and another write item sets the time of lease.
- Acquire multiple leases atomically
  - Same as above, except that there are multiple compare and write items. Note that each lease can be in a different memory node.
- Change data if lease is held
  - A compare item checks that a lease is held and, if so, write items update data.

Bibliography - 1

- Marcos Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch and Christos Karamanolis, Sinfonia: A New Paradigm for Building Scalable Distributed Systems, SOSP '07.
- M. Alrifai, P. Dolog, Balke Wolf-Tilo and W. Nejdl, Distributed Management of Concurrent Web Service Transactions, IEEE Transactions on Service Computing, Vol. 2, No. 4, pp. 289-302,2009.
- S. Bhiri, O. Perrin and C. Godart, Ensuring Required Failure Atomicity of Composite Web Services, WWW 2005, pp. 138-147, 2005.
- L.F. Cabrera, G. Copeland, T. Fruend, J. Klein, D. Langworthy, F. Leymann, I. Robinson, T. Storey and T. Thatte, Web Services Business Activity Framework, 2005, http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/WS-BusinessActivity.pdf
- P. K. Chrysanthis, K. and Ramamritham, A Formalism for Extended Transaction Models, Proc. of 17th International conference on Very large databases, pp. 103-112, 1991.
- Sudipto Das, Divyakant Agrawal and Amr El Abbadi, ElasTras: An Elastic Transactional Data Store in the Cloud, http://www.usenix.org/event/hotcloud09/tech/full_papers/das.pdf

Bibliography - 2

- H. Garcia-Molina. and K. Salem, SAGAS, Proceedings of ACM SIGMOD Conference on Management of Data, 1987.
- P. Grefen, J. Vonk and P. Apers, Global transaction support for workflow management systems: from formal specification to practical implementation. The VLDB Journal 10, pp.316-333, 2001.
- P. Michael Melliar-Smith and Louise E. Moser, Achieving Atomicity for Web Services Using Commutativity of Actions, Journal of Universal Computer Science, 13(8), pp. 1094-1109, 2007.
- F. Montagut and R. Molva, Augmenting Web Services Composition with Transactional Requirements, Proc. IEEE Int. Conf. on Web Services (ICWS 2006).
- J.E.B. Moss, Nested Transactions: An Approach to Reliable Distributed Computing, MIT/LCS/TR-260, Laboratory for Computer Science, Massachusetts Institute of Technology, U.S.A, 1981.
- M. Papazoglou, Web Services and Business Transactions, World Wide Web: Internet and Web Information Systems, 6, pp. 49-91, 2003.
- C. Pu, G. Keiser and N. Hutchinson, Split-Transactions for Open-Ended Activities, Proceedings of the 14th International Conference on VLDB, 1998.

Bibliography - 3

- H. Schuldt, G. Alonso, C. Beeri and Schek Hans-Jorg, Atomicity and Isolation for Transactional Processes, ACM Transactions on Database Systems, 27(1), pp. 63 – 116, 2002.
- D. Skeen and M. Stonebraker, A Formal Model of Crash Recovery in a Distributed System, IEEE Transactions on Software Engineering, 9(3), pp. 219-228, 1983.
- K. Vidyasankar, Serializability, Encyclopedia of Database Systems, Editors-in-chief: Liu Ling; Ozsu, M. Tamer, Springer 2009.
- K. Vidyasankar. and G. Vossen, Multi-Level Modeling of Web Service Compositions with Transactional Properties, Journal of Database Management, Special issue on Service-oriented Computing.
- K. Vidyasankar and G. Vossen, A Multi-Level Model for Web Service Composition, Proceedings of International Conference on Web Services (ICWS 2004), San Diego, July 6-9, pp. 462-469, 2004.
- K. Vidyasankar, P. Radha Krishna and Kamalakar Karlapalem, A Multi-level Model for Activity Commitments in E-contracts, 15th Int. Conf. in Cooperative Information Systems (CoopIS 2007), Vilamora, Protugal, LNCS Vol. 4803, Springer 2007, pp. 300-317, 2007.
- K. Vidyasankar, P. Radha Krishna and Kamalakar Karlapalem, Study of Execution Centric Payment Issues in E-contracts, Proc. IEEE Services Computing Conference (SCC), Hawaii, Vol. 2, pp. 135-142, 2008.

Bibliography - 4

- K. Vidyasankar, P. Radha Krishna and Kamalakar Karlapalem, Study of Dependencies in Executions of E-contract Activities, 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009), Riga, Latvia, September 2009, LNCS Vol. 5739, 2009, pp. 301-313, 2009.
- Zhang Liang-Jie, Zhang J. and Cai H., Services Computing, Springer and Tsinghua University Press, 2007.
- Ting Wang, Paul Grefen and Jochem Vonk, Abstract Transactional Construct: Building a Transaction Framework for Contract-Driven, Service-Oriented Business Processes, ICSOC 2006, LNCS Vol. 4294, Springer, pp. 434-439, 2006.
- Zhou Wei, Guillaume Pierre and Chi-Hung Chi, Scalable Transactions for Web Applications in the Cloud, http://www.globule.org/publi/STWAC_europar2009.pdf
- OASIS Committee Specification, "Business Transaction Protocol," version 1.0, May 2002.
- OASIS Web Service Atomic Transaction (WS-AtomicTransaction), 2007, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf
- OASIS Web Service Business Activity (WS-BusinessActivity), http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.pdf
- OASIS Web Service Coordination (WS-Coordination), 2007, http://docs.oasis-open.org/ws-tx/wscoor/2006/06.

Bibliography - 5

- OASIS Committee Specification, “Business Transaction Protocol,” version 1.0, May 2002.
- OASIS Web Service Atomic Transaction (WS-AtomicTransaction), 2007, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf
- OASIS Web Service Business Activity (WS-BusinessActivity), http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.pdf
- OASIS Web Service Coordination (WS-Coordination), 2007, http://docs.oasis-open.org/ws-tx/wscoor/2006/06.

Thank you

Payments in E-Contracts

Payment issues

- The vital issue of payments in e-contracts are the following.
  - Payments are made to parties.
  - They may be constrained by clauses.
  - Most importantly, they are meant for, and so are closely related to, the execution of activities in the contract.
- We can identify the execution states of the activities in terms of their transactional (compensatability, retriability and commit) properties, and relate the states to costs of, and payments for, the activities.

Two aspects of payments

- First, one should be able to ascertain that the activities have been executed satisfactorily to deserve payment.
- Second, the amounts of payments need to be determined.
- Both these require a good understanding of the execution states of the activities and hence the execution state of the e-contract.

Payments

- Two aspects – enabling and making payments.
- Payment options include the following:
  - For each activity, either a single payment or multiple (partial) payments may be enabled at various states of execution.
  - Payments can be made once or in several installments. The installments need not correlate to the enabling points.
  - A payment can be fully or partially refundable, or non-refundable.
  - Payment for an activity may be made ahead of its execution or after the execution. As stated earlier, the actual cost and hence the actual amount to be paid may be known only at the end.
- A payment monitoring system should keep track of the state of termination, payment-enabled and payment-made points and the amounts, for each activity.

Cost and payment

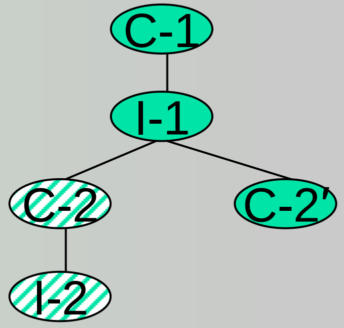
- Let C be a composite activity consisting of basic activities a1, a2, etc.
- There are two aspects of the cost of execution of ai:
  - for ai and
  - for C, that is, the cost charged to C and hence to be paid by (the service executing) C to (the service executing) ai.
- We denote the first as cost(ai) and the second as payment(ai).

Calculating cost and payment

- A cost is associated with an s-termination of an activity. Different s-terminations may cost different amounts. (Example: Non-refundable flight ticket may cost more.)
- An activity ai that is not executed may cost nothing. If executed but compensated, cost(ai) may be non-zero, but payment(ai) may be zero.
- Each re-execution may incur additional cost. Therefore, the final value of cost(ai) may be known only at the end of the execution.
- If re-execution costs are not charged to C, then payment(ai) may be known on weak commitment of ai.

Payment and Cost - 1

- Each activity in the execution-tree has to be paid for.



- Payment(s) may be enabled and made in any of the states of execution of that activity, and also in the states of weak and strong commitments relative to C.
  - For example, payment for ai may be enabled either when its execution is locally weakly committed or only when it is weakly committed relative to C, meaning that it will not be compensated even by a compensating activity in C.

Payment and Cost - 2

- If an execution ai is compensated by execution ai′ of a compensating activity, then both ai and ai′ will appear in the execution-tree, and costs may be attributed to them individually.
- Similarly, if retrying of ai is done by executing additional activities, their executions will also be in the execution-tree and costs can be assigned to them.

Payment and Cost – 3

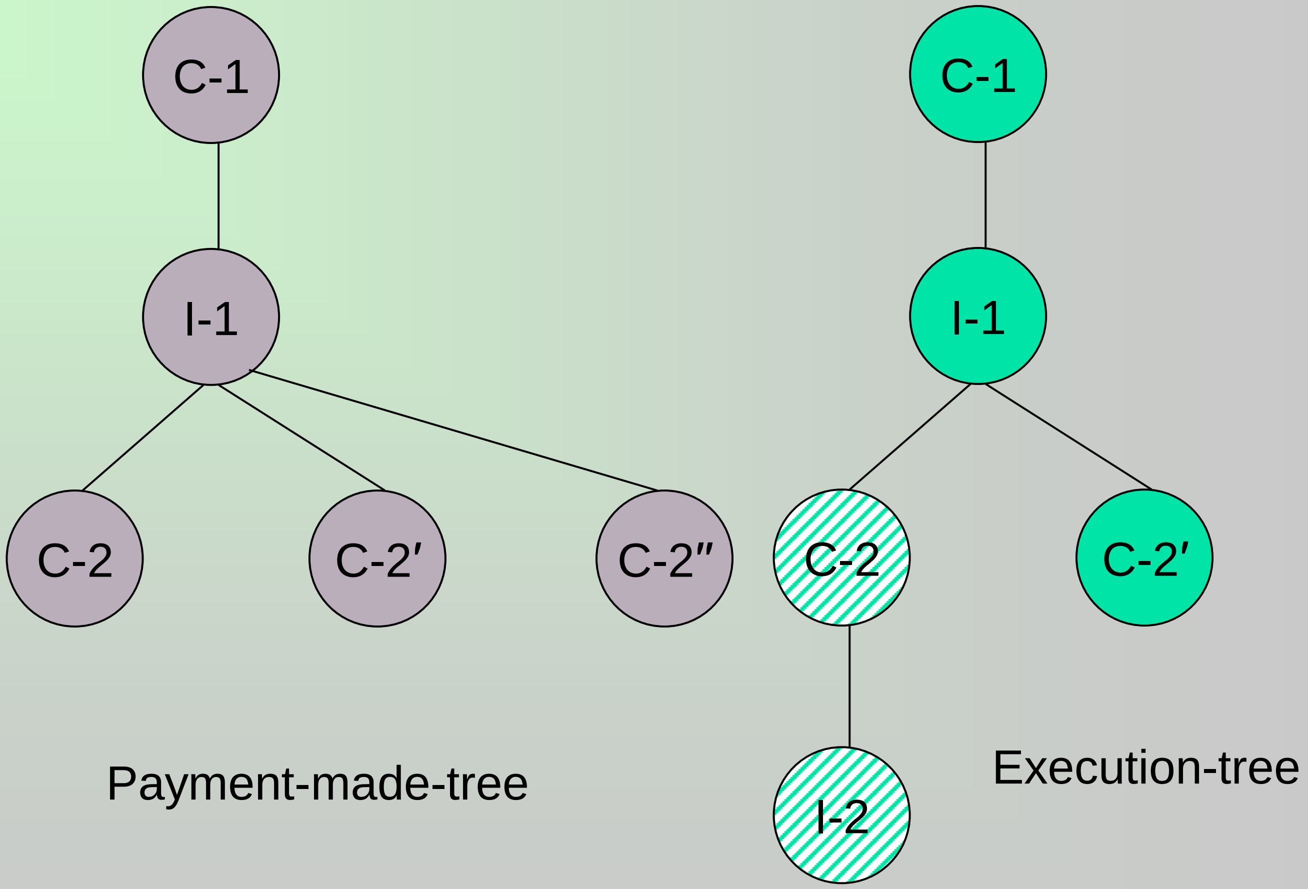
- Enabling and making payments for different activities may occur at different times.
- Dependencies may exist between enabling/making payments for different activities.
- Dependencies may also exist between enabling/making payments for one activity and starting the execution (similarly, compensating, weakly committing and strongly committing) another activity, and vice versa.

Payment Trees

- At any stage, the activities whose payments have been enabled and those whose payments have been made can be kept track of with a payment-enabled-tree and a payment-made-tree, respectively.
- Note that the execution-tree and the two payment trees are all sub-trees of the composition graph **C**. As the execution of the contract progresses, all the three trees will grow. By comparing them, the correspondence between the execution of the activities and enabling/making payments for them can be obtained.

Execution example – cont’d

- Here is a payment-made-tree for the previous example.
- Payment for I-2 has not been done yet.
- Payment for C-2″ has also been done even though only one of C-2′ or C-2″ is to be executed. The payment for non-executed activity needs to be adjusted later on.



Multi-level Model Transactional Properties and Payments

- Enabling and making payments can be tied to the transactional properties defined relative to different ancestors.
    - For example, payment for a-i can be enabled only when it is weakly committed relative to its grand-parent U. This may be appropriate when payment authorizations come from U and not from (parent) C.

Payment Issues Discussion - 1

- We have expressed the property that costs are determined by the executions. It is also possible that costs and payments influence the executions.
  - We associated a cost for each re-execution. Then, the total cost for execution of an activity will increase with the number of re-executions. If a maximum cost is stipulated for an activity, then that could limit the number of re-executions.
  - Payments may influence the time of commitment. For example, a non-refundable payment can be associated with weak commitment which can be delayed until it is certain that the execution does not need to be compensated. Similarly, if no retrys are expected after payment, then strong commitment can be combined with the payment.

Payment Issues Discussion - 2

- As mentioned earlier, activities in e-contract may be executed autonomously.
- Details of payments for them may also be kept autonomously.
- The ability to deal with payment trees of different levels, with activities described at different depths of the hierarchy, supports the autonomy.

Open Issues

- In the literature, the inter-dependency among contract satisfaction, activity execution and payments has not been explicitly modelled. The utility of such modeling in deploying and managing the commitment and payment aspects of e-contracts is immense.
- Some open issues are:
  - Initiating payment transactions for making appropriate payments;
  - Extraction of related clauses for payments and monitoring of payments; and
  - Finding profitable contracts in an organization when multiple contracts are in execution.