Memorial
University of Newfoundland

Computer Science

TECHNICAL REPORT #2010-01

On E-contract Activity Commitments

by

K. Vidyasankar

In conjunction with:
P. Radha Krishna
SET Labs, Infosys Technologies Limited
Kamalakar Karlapalem
International Institute of Information Technology

Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada A1B 3X5

May 2010

On E-contract Activity Commitments

K. Vidyasankar [&] [1], P. Radha Krishna* and Kamalakar Karlapalem[+]

[&]Department of Computer Science, Memorial University, St. John's, Canada, A1B 3X5.
vidya@mun.ca
*SET Labs, Infosys Technologies Limited, Hyderabad, India.
Radhakrishna_p@infosys.com
[+]International Institute of Information Technology, Hyderabad, India.
kamal@iiit.ac.in

Abstract

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 the parties satisfying the clauses, with the associated terms of payment. The activities in a contract are generally complex and interdependent. They may be executed by different parties autonomously and in a loosely coupled fashion. They may be compensated and/or re-executed at different times relative to the execution of other activities**.** Both the initial specification of the activities and the later verification of their executions with respect to compliance to the clauses are tedious and complicated. We believe that 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. The goals of an e-contract include precise specification of the activities of the contract, mapping them to deployable workflows, and providing transactional support for their executions. Towards facilitating this, we present a multi-level composition model for activities in e-contracts. Our model allows for the specification of a number of transactional properties, like atomicity and commitment, for activities at all levels of the composition. It enables the study of the interdependencies between the executions of e-contract activities. This will help in monitoring behavioral conditions stated in an e-contract during its execution. We show also that the transactional properties facilitate computing the cost of execution of the activities and coordinating payment.

1. Introduction

An *electronic contract*, or *e-contract* in short, 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 the parties satisfying the clauses, with the associated terms of payment.

Consider, for example, a contract for building a house. The parties of this contract include a customer, a builder, a bank and an insurance company. The contract has several parts: (a) The builder will construct the house according to the specifications of the customer. Some of the activities such as carpentry, plumbing and electrical work may be sub-contracted; (b) The customer will get a loan for the construction from the bank. He

[1] This research is supported in part by the Natural Sciences and Engineering Research Council of Canada Discovery Grant 3182.

will apply for a mortgage, and work out details of payment to the builder, directly by the bank, after inspection of the work at multiple intervals; and (c) The house shall be insured comprehensively for the market value covering fire, flood, etc. in the joint names of the bank and the customer. The activities of the customer and the builder include the following.

- Customer: (i) submitting the loan application, (ii) setting up coordination between bank and builder, (iii) receiving payments and (iv) arranging monthly repayments.
- Builder: (i) scheduling different works involved in the construction, (ii) procuring raw material, (iii) building the house as per the agreement, (iv) giving part of the work to sub-contracts, if any, (v) receiving the payments, (vi) making payments to its staff and sub-contract parties, if any, and (vii) handing over the constructed house to the customer.

An example of a clause relating to payments can be, in verbatim, as follows.

"*If the bank is of the opinion that the progress of work of construction of the said house is unsatisfactory, the bank shall be at liberty to decline to make payment of any not-yet-disbursed installment of the said loan or at its discretion postpone the payment thereof until such time the bank is satisfied that the cause or causes for its dissatisfaction with the progress and quality of work has or have been removed.*".

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 due, partly, to the complexity of the activities. Typically, the activities are interdependent with other activities and clauses. They may be executed by different parties autonomously, in a loosely coupled fashion. They are long-lasting. Though the desirable outcomes of their executions are stipulated in the contract specification, their executions may yield unexpected results. This might result in re-design and even re-specification of the contract. We assert that a key to handle the complexity in executions of contract activities is adherence to transactional properties.

In database applications, *atomicity* is strived for in a (simple) transaction execution. That is, a transaction is executed either completely or (effectively) not at all. Given a non-null partial execution, the former is obtained by *forward-recovery* and the latter by *backward-recovery*. On successful completion, the transaction is *committed*. In multi-database and other advanced database applications, transactions may be committed (locally) and then rolled back logically, by executing compensating transactions. This property is called *compensatability.* The property of repeatedly executing a transaction until successful completion is also considered; this is called *retriability.*

In e-contract activities also, both compensatability and retriability properties are encountered for the activities, and in fact, in more sophisticated ways. For example,

(i) Both complete and partial executions may be compensated,
(ii) Both successful and unsuccessful executions may be compensated,
(iii) Even "committed" executions may be retried,
(iv) Retrying may mean, in addition to re-execution, "adjusting" the previous execution, and
(v) Activities may be compensated and/or retried at different times, relative to the execution of other 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.

In this paper, we propose a *multi-level composition model* for activities in e-contract. We start with basic activities and construct composite activities hierarchically. 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 activity will correspond to the "single" activity for which the contract is made. We call this the *contract-activity*. (We note that there could be multiple contracts for a single activity. For example, for building a house, there could be separate contracts between (i) customer and the builder, (ii) customer and the bank, (iii) customer, bank and insurance company, etc. These contracts will be related. We consider this set of contracts as a part of a single high level contract whose contract-activity is building the house.) Then, our contention is that the execution of each activity, at every level, should satisfy transactional properties.

Payments are made to parties. They may be constrained by clauses. Unlike in traditional information systems, executions of activities in e-contracts are subjected to risks and losses (in case of non-performance), trust issues (among parties with respect to satisfactory execution), ambiguity in specifications (in clauses), different types of failures (especially of non-electronic ones) and potential variations in outcomes. All these parameters influence the cost of an activity in the e-contract. Most importantly, payments are meant for, and so are closely related to, the execution of activities in the contract. We show that our multi-level composition model helps in computing the costs and for monitoring payments.

Every activity in the contract must be *closed* at some time. On closure, no execution related to that activity would take place. The closure could take place 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.* The e-contract closure is mostly a human decision. It may involve auditing, handing over documents, releasing assets, dispute resolution (if any), settling payments (including post-deliverable payments), etc. However, in this work, we consider commitment of e-contract as e-contract closure. We use the term *e-contract commitment logic* 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.

In e-contracts, interaction occurs between parties which are autonomous and work together using loosely-coupled services. A contract consists of numerous activities that are to be carried out by parties and contract clauses that address a specific concern in the business interaction. Since inter-organizational work elements are handled through contracts and most of the contracts are complex and voluminous, manual verification is both expensive and error prone. This necessitates a well-defined commitment framework for correctness and successful execution of e-contracts [3, 12].

1.1. Architecture

Transactional semantics, workflow semantics, clauses and payment components of e-contract need to be considered for addressing e-contract commitments. Workflow semantics deals with the composition logic, namely, the semantics of the executions of the individual activities that constitute the workflow. Transactional semantics deals with the commitment logic, about atomicity, forward- and backward-recovery and commitment of the executions, and closure of the activities and the e-contract. Both clauses and payments influence, and are influenced by, both the workflow and transaction semantics.

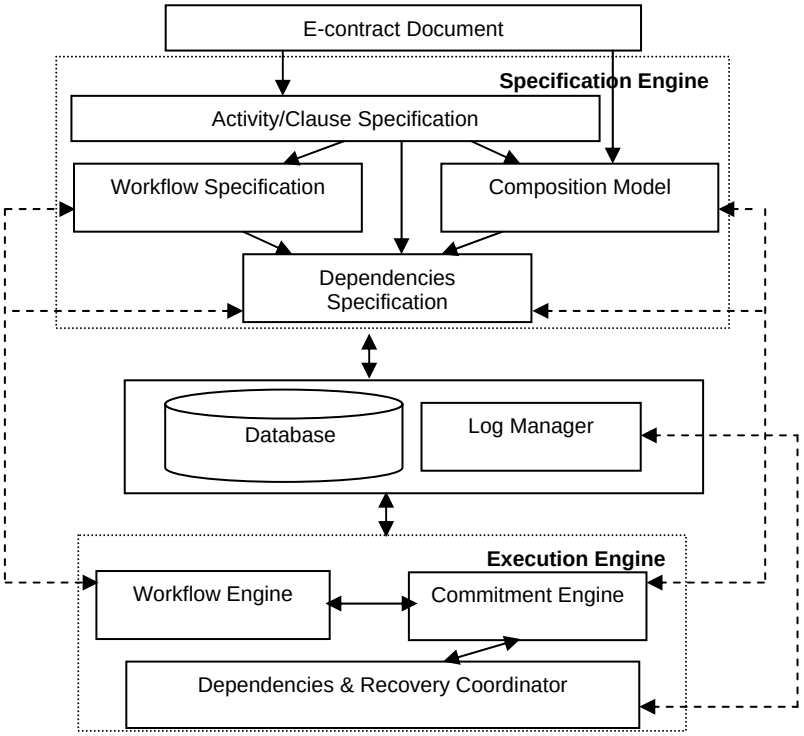

Figure 1. E-Contract Activity Commitment System - High level view

Figure 1 shows a high-level view of activity commitment system. The figure has two components: specification engine and execution engine. E-contract document is the basic input to the entire system. The specification engine extracts activities and clauses specifications. These specifications are useful to generate workflow specifications and multi-level composition model. The e-contract activity characteristics described above give rise to sophisticated interdependencies between executions of different activities. These dependencies deeply impact both the recovery and commitment aspects. Activity and clause specifications are also useful to derive the dependencies between activities. Using the audit trials provided by the log manager, the components of the execution engine ensure the atomicity of the executions of the e-contract activities.

In this paper, we focus on execution engine, particularly on the aspects required in developing commit design and dependencies and recovery coordinator components.

1.2. Contributions and Organization of the Paper

In this paper, we propose a multi-level composition model for the (composite) activities of an e-contract. Transactional properties have been defined to suit the real world, non-electronic, activities. The salient points are the following.

i. Transactional properties are defined for executions of activities rather than activities themselves. This accounts for the fact that different executions of the same activity might have different characteristics.
ii. Atomicity is defined for executions of composite activities of any level in spite of the executions of even some basic activities being non-atomic. This helps in dealing with backward- and forward-recoveries at each level independent of its descendent levels.
iii. The scope of retriability is extended from executing the same activity again, or executing some other substitute activity, to adjustments to the original execution.
iv. Two levels of commitment, weak and strong, are defined. On weak commitment, the execution becomes non-compensatable, and on strong commitment it becomes non-retriable. Weak commitment is the commitment property of the traditional database operations and the pivotal property of multi-database operations. The strong commitment property definition is new.

Both (a) defining transactional properties for activities of a contract and (b) influencing e-contract design with transactional properties are novel and have not been done before.

We use the composition model to study the interdependencies among the executions of the activities, and also the dependencies between the executions and the payments for the activities. We consider (i) the payment amount for the execution of an activity, (ii) the time of payment relative to the execution and (iii) tracking the payment against the execution of the activity.

The rest of the paper is organized as follows. Some related work is described in Section 2. We present the basic concepts related to our model in Section 3 and the model in Section 4. Payment issues are considered in Section 5. Section 6 concludes the paper.

2. Related Work

Considerable work has been carried out on the development of e-contract framework and architectures, commitment and monitoring of e-contracts. We cite some of them in the following.

Chiu et al. [6] presented a meta-model for e-contracts and templates, an architecture and a methodology for developing e-contract enforcement rules. The CrossFlow project [11] introduces dynamic contracting and configuration for service enactment and defines inter-organizational business process among the parties. SweetDeal system [13] allows software agents to create, evaluate, negotiate and execute e-contracts with substantial automation and modularity. E-ADOME [15] and CrossFlow [16] systems describe the workflow interfaces as activities and transitions in e-contracts. In the same way, Chiu et al. [5] develop a framework for workflow view based e-contracts for e-services. Grefen and Vonk [10] describe the relationship between transaction management systems and workflows for transactional business process support. Wang et al [27] describe a Business

Transaction Framework based on Abstract Transactional Constructs, which provides a specification language for identifying and interpreting clauses in e-contracts.

Krishna et al. [17] consider activity-party-clauses and activity-commit diagrams for modeling and monitoring e-contracts. These constructs are used to express the execution order and execution status of the contract that is being considered. Rouached et al. [19] present an event-based framework associated with a semantic definition of the commitments expressed in the event calculus to model and monitor multi-party contracts. Jain et al. [14] present a flexible composition of commitments, known as metacommitments. These commitments are mainly associated with the role of a party and ensuring whether a particular activity is committed or not. They do not refer the commitments with respect to the execution states of an e-contract activity.

Xu [28] proposes a pro-active e-contract monitoring system that is based on contract constraints and guards of the contract constraints to monitor contract violations. This paper represents constraints using propositional temporal logic in order to provide formal semantics for contract computation at the contract fulfillment stage. However, the formalism in this paper does not provide the execution level semantics of an e-contract commitment. Farrell et al. [9] present automated performance monitoring of e-contracts, in terms of tracking contract states by expounding an XML formalization of the event calculus and ecXML. A rule-based approach is presented in [13] to deal with exceptions raised during e-contract execution.

Transaction concepts, as the ACID (Atomicity, Consistency, Isolation and Durability) properties, were first proposed for database applications. In early applications, the database system was centralized, the execution time was relatively short, and the number of concurrent transactions was relatively small. For distributed database systems and long-running transactions, several variations of the basic transaction model were proposed. Some examples are chained transactions, saga, nested transactions and ACTA framework [7, 8]. Later proposals include [4] for long-running transactions and [1] for workflows. The Activity-Transaction Model (ATM) in [2] allows long-running transactions and provides recovery mecahnisms for transaction workflows that consist of transaction hierarchies. Papazoglou [18] describes a taxonomy of e-business transaction features and presents a business transaction model that relaxes isolation and atomicity requirements of ACID transactions in a loosely coupled environment consisting of autonomous trading partners. This paper also describes backward- and forward-recovery for long-running business transactions.

Compensatability and retriability properties were first identified in the context of atomicity of multi-database applications (for instance, [21]). To achieve atomicity (of a global transaction) in autonomous execution (of the subtransactions), a multi-database transaction is modeled to consist of a sequence of compensatable transactions, followed possibly by a *pivotal* (that is, non-compensatable) transaction and a sequence of retriable transactions. In particular, each multi-database transaction can have at most one pivot. Schuldt et al. [20] extended this idea to transactional processes by allowing multiple pivots. Clearly, with multiple pivots, atomic execution may not be possible (when some pivots are executed but others cannot be executed). They defined a property, called *guaranteed termination,* which formalized "graceful" termination of the transaction after some pivots were executed. In addition, the pivots in a guaranteed termination were executed in sequence. Further extension was done in [22, 23], in the context of

composition of Web Services. In this work, (i) the guaranteed termination concept was extended to atomicity (of global transaction, or composite activity or service), (ii) forward- and backward-recovery procedures for achieving atomicity were given, and (iii) non-sequential, tree-like, execution of the pivots was accommodated. Then the transactional properties (atomicity, compensatability, retriability and pivot) were extended to hierarchically composed activities/services. It was shown that the transactional properties can be considered at each level independently of the properties of the other level activities. The proposal to address activity commitments in e-contracts in this work is along the lines of [24, 25, 26] but tailored and extended to e-contract environment.

3. Basic Concepts

In this section, we present the concepts and notations relevant for transactional properties in the context of e-contracts, and in the next section we present our model.

Basic Activities

We consider certain activities as *basic* in our model. Typically, these are the activities which cannot be decomposed into smaller activities, or those that we want to consider in entirety, and not in terms of its constituent activities.

In e-contract environment, some basic activities may be executed 'electronically' (for example, processing a payment), but most others will be non-electronic (for example, painting a door). We desire that all basic activities are executed atomically, that is, it is either not (effectively) executed at all or executed *completely*. However, incomplete executions are unavoidable and we consider them in our model.

Constraints

Each activity is executed under some constraints. Examples of constraints are (i) who can execute the activity, (ii) when it can be executed, (iii) whether it can be executed within a specified time period, (iv) cost of execution, (v) what properties need to be satisfied for an execution to be acceptable, and (vi) compensatability or other transactional properties. The first four constraints relate to workflow semantics. The last two relate to transactional semantics.

A complete execution of an activity that satisfies all the constraints specified for the execution of that activity *at the time of its execution* is called a *successful termination*, abbreviated *s-termination*, of that activity. The constraints themselves are specified in terms of an *s-termination predicate*, or simply, *st-predicate*. A complete execution which does not satisfy the st-predicate is called a *failed termination*, abbreviated *f-termination*. The s- and f-termination distinction is applied to incomplete executions also, depending on whether the st-predicate is satisfied thus far.

Example 1: Consider the activity of painting a wall. The execution is incomplete while the wall is being painted, and complete once the painting is finished. If the paint job is good at the end (respectively, in the middle), the execution is a complete (respectively, incomplete) s-termination.

If the paint job is not satisfactory, we get a complete or incomplete f-termination. The st-predicate specifying the goodness of the job could be: (i) one undercoat and one other coat of paint and (ii) no smudges in the ceiling or adjacent walls.

For many activities, especially non-electronic ones, some acceptability criteria may be highly subjective and may depend on the application environment. For example, consider the activity of building a wall. Quantitative aspects such as the dimensions of the wall, its location, etc. can be expressed easily. Smoothness of the finished surface and extent of the roundedness of the corners will be application dependent. The requirements for a wall in a children's hospital will be different from those for one in an army barrack. We propose that a predicate, termed *property-predicate*, be defined for each of the requirements and the acceptability, that is the st-predicate, be stated in terms of satisfying a Boolean expression of the property-predicates. Determining whether a property-predicate is satisfied or not in an execution will be left to the application semantics. Thus, the st-predicate for the construction of a wall could be *(d AND s AND r)* where d is the dimension predicate stating whether the dimensions of the wall are according to specifications, s is the smoothness predicate and r is the roundedness (of the corners) predicate. Then, an execution which does not satisfy one or more of these predicates will be an f-termination. Clearly, several different f-terminations are possible. As another example, the st-predicate for finishing a wall could be *((u AND o) OR (u AND w))* where u refers to an undercoat of painting, o is an overcoat with smooth finish and w is wall-papering. Here, two s-terminations are possible, one yielding a painted surface and the other with wall paper.

The constraints may change, that is, the st-predicate of an activity may change, as the execution of the contract proceeds. In the above example of building a wall, the required thickness of the wall may change from 6 inches to 8 inches, thus changing the dimension predicate. Similarly, two coats of paint may be required in addition to undercoat. Such changes may invalidate a previous s-termination. When this happens, the execution needs to be adjusted. We note also that, with changes in the st-predicate, an earlier f-terminated execution may become an s-termination. *It follows that we may not know whether a termination is an s-termination or an f-termination until some time later.*

Compensatibility

One of the ways an execution can be adjusted is by compensation, namely, nullifying the effects of the execution. Absolute compensation may not be possible in several situations. In some cases, the effects of the original execution may be ignored or penalized and the execution itself considered as compensated. It is possible that an execution can be compensated within a certain time, but not afterwards. The time could be "real" time (for example, flight reservations can be cancelled without penalty within 24 hours of booking, and vinyl flooring glued to the floor can be removed before the glue sets) or specified relative to the execution of some subsequent activities (for example, flight bookings can be cancelled until paid for, and a (stolen) cheque can be cancelled before it is cashed). Inability to execute a compensating activity within a prescribed time limit may also make the original execution non-compensatable.

Note that we do not attribute compensatability property to an activity, but only to an execution of that activity. For the same activity, some executions may be compensatable,

whereas others may not be. For example, when we book flight tickets we may find that some tickets are non-refundable, some are fully refundable, and some others partially refundable. Purchasing a fully refundable ticket may be considered to be a compensatable execution, whereas purchasing any other type of ticket could be non-compensatable. Thus, compensatability of the execution (purchasing a flight ticket) is known only during execution, and not at the specification time of the activity. We look at *compensation as a logical roll back of the original execution*. Then, compensation may also be done by executing some other, *compensating*, activity. The compensating activity could be executed at different levels, in our multi-level model.

Retriability

Another way of adjusting an execution is by *retrying*. By retriability, we mean the ability to get a complete execution satisfying the (possibly new) st-predicate. It is possible that the original execution is compensated fully and new execution carried out, or the original execution is complemented, perhaps after a partial compensation, with some additional execution, for instance, the second coat of painting in Example 2. Another example is, a day after pouring concrete for the foundation of a house, the thickness of the concrete may be found to be insufficient, and additional concrete poured for the required thickness.

Retriability may also be time-dependent. It may also depend on the properties of execution of other preceding, succeeding or parallel activities. Again, in general, some executions of an activity may be retriable, and some others may not be retriable.

We note that retriability property is orthogonal to compensatability. That is, an execution may or may not be retriable, and, independently, may or may not be compensatable.

Execution States

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

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, and we take this as an f-termination, or
    b. re-executable to get eventually a complete s-termination, and we identify this state as non-compensatable but retriable.
3. Continuing re-executions in state 2.b, at some stage, we get a complete s-termination which is non-compensatable and non-re-executable.

It is also possible that an (un-compensated) execution remains in state 1 and never goes to state 2, and similarly an execution is in state 2.b, but never goes to state 3

We say that an execution in state 2.b is *weakly committed*, that is, when it is or has become non-compensatable, but is retriable. An execution in state 3 is *strongly committed*. We note that both weak and strong commitments can be forced upon externally also. That is, the execution can be *deemed* as (weakly or strongly) committed, for reasons outside of that execution. An example is payment to a sub-contractor for execution of an activity, and the non-obligation and unwillingness of the sub-contractor to compensate (in case of weak commitment) or retry (in case of strong commitment) the execution after receiving the payment. We say also that an activity is *weakly* (*strongly*) *committed* when an execution of that activity is weakly (strongly) committed.

We allow compensatability and retriability properties to be applicable to incomplete executions also. We assume the first two of the above state transition sequences for partial executions. That is, a partial execution is both compensatable and retriable in the beginning, and may become non-compensatable at some stage. Then, if it is retriable, that is, a complete s-termination is guaranteed, then the execution can be weakly committed. Note that we are simply allowing the transition from uncommitted to weakly committed state to occur even before the execution of the activity is complete. We do not allow transition from weakly committed to strongly committed state until (or some time after) the execution is completed.

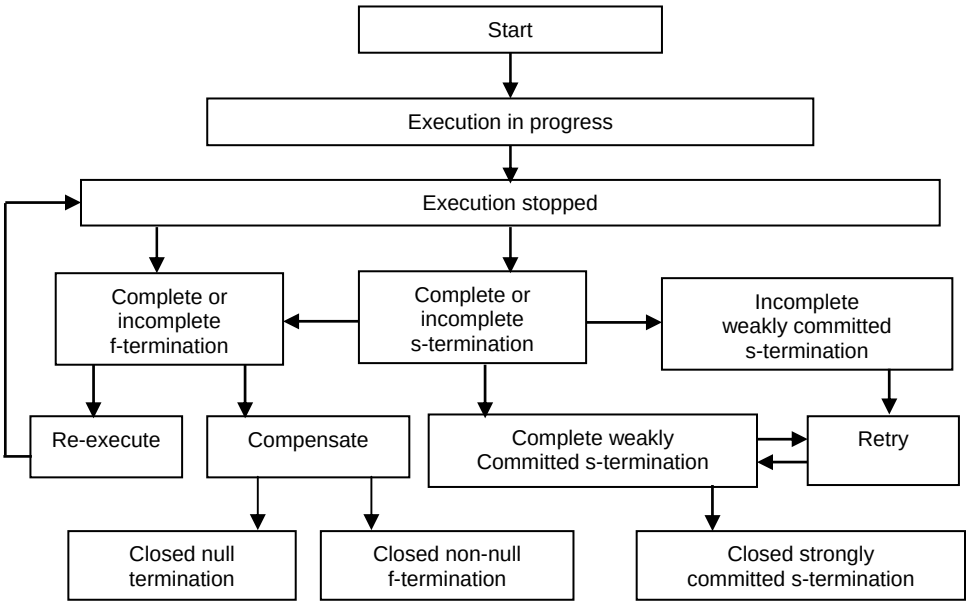

Figure 2. Execution stages of an activity

Figure 2 depicts the execution stages (boxes) of an activity, and possible transitions (arrows) between them. Some notable points are the following.

- Re-execution may possibly occur after a partial or full backward-recovery.
- As stated earlier, retry denotes re-execution that is guaranteed to yield an s-termination.
- A full backward-recovery yields the null termination. If re-execution of the activity is intended after the null termination, we take the backward-recovery as part of retry; otherwise, it is taken as compensation.

- A complete s-termination may become an f-termination, with a change in st-predicate. If this happens before weak commitment, the transitions of an f-termination are followed. Otherwise, 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.

The "final" state of execution of a basic activity is closure. Figure 2 shows three possible states of closure: (i) null; (ii) non-null (incomplete or complete) f-termination; and (iii) (complete) s-termination, which also corresponds to strong commitment of the execution.

Figure 2 is applicable to composite activities also. Complete and incomplete, and s- and f-terminations can be defined for composite activities, analogously. This is done in the model. We explain this later.

We illustrate the different categories with the following example.

Example 2: Let *U* be a composite activity consisting of (i) writing and printing a letter, (ii) preparing an envelope, and (iii) inserting the letter in the envelope and sealing it. Call the activity (ii) as *C*. Then *C* is composed of (a_1) printing the From and To addresses on the envelope, perhaps with a printer and (a_2) affixing a stamp on the envelope. Consider an execution of *U*. The following possibilities arise.

- (i) is done but (ii) fails possibly because of printing a wrong address. Now we may decide not to bother preparing a new envelope and sending the letter. This is an incomplete f-termination.
- (i) and (ii) are done. (iii) is not done (yet). This is an incomplete s-termination.
- All the three activities are done, but we realize afterwards that the address is wrong, that is, (ii) is not executed correctly. This is a complete f-termination.
- All activities have been done correctly. This is a complete s-termination.

Different terminations in an execution of an activity are given in Figure 3. In the case $m > 1$, each Termination-i, for i between 1 and m-1, is both compensatable and re-executable. Termination-m either leads to a (compensated or non-compensated) f-termination or becomes a weakly committed wc-termination-1. In the latter case, we eventually get a strongly committed sc-termination. Note that the case where both m and n are 1 refers to the first termination itself being successful, and weakly and strongly committed.

4. Composition Model for Activities

We now describe our composition model for the activities in an e-contract. We start with a specification of *one* level, the "bottom" level, in the first two sub-sections, and give multi-level model in Section 4.3.

4.1. Path Model

We start with a simple model, called the *path model*, to illustrate the various key aspects. We will extend it to a general model in the next sub-section. Our description is in four parts – composition, execution, transactional properties, and implementation details. We use bold font to denote compositions, and italics to denote their executions, that is, the composite activities.

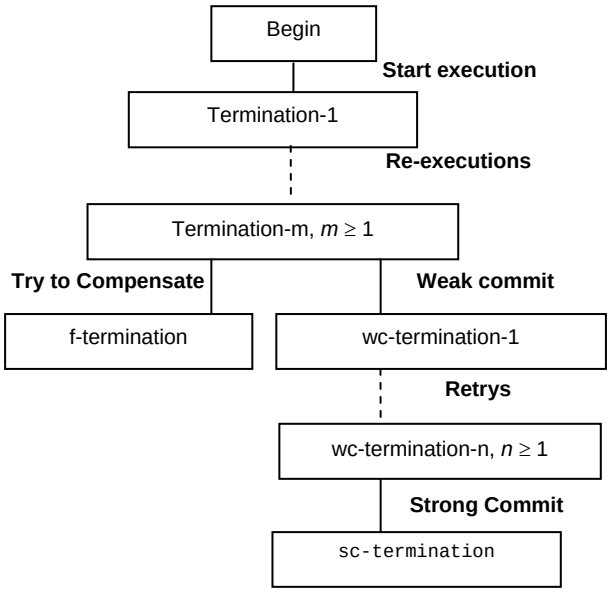

Figure 3. Different terminations

A. Composition

- Composition **C** is a rooted tree. It is for an activity of a higher level composition **U**.
- An st-predicate is associated with **C**. This will prescribe the s-terminations of **C** (We define s-terminations of a composition later).
- Nodes in the tree correspond to basic activities. They are denoted as $\mathbf{a_1}$, $\mathbf{a_2}$, etc.
- With each node in the tree, an st-predicate and a *children execution predicate*, abbreviated *ce-predicate*, are associated.
- The st-predicate specifies s-terminations of that activity. The ce-predicate specifies, for *each* s-termination of that node, a set of children from which *exactly one* child has to be executed, the child being chosen according to a given partial order of preferences. The ce-predicates for the leaf nodes of the composition are null.
- We assume that the st-predicate and ce-predicate of each of the nodes in **C** are derived from the st-predicate of **C**.

Example 3: Figure 4 (a) shows a composition where C_i's are construction activities for a product and I_j's are Inspection activities. After the first two stages, C_0 and C_1, of the construction, the inspection I_1 is carried out. Depending on the result, say quality of the product after C_1, C_2 is carried out if possible, and C_2' or C_2'' otherwise, in that order. This will be the ce-predicate at I_1. Only the inspection I_2 after C_2 is shown. The st-predicate for each C_i will be the guidelines to be followed for that construction. The st-predicate for each I_i will be the acceptable results of the things to be checked in that inspection.

B. Execution

- An execution of activity $\mathbf{a_i}$ is denoted a_i.
- A successful execution *E* of **C** yields a composite activity *C*. The execution consists of s-terminations of activities in the path from the root to a leaf (and f-terminations of some other activities). The corresponding nodes form a sub-tree of **C,** called *execution-tree*. If all the activities in this path have been executed completely, then *E* is a *complete* execution of **C**. (The example, shown in Figure 4(b), has executions of

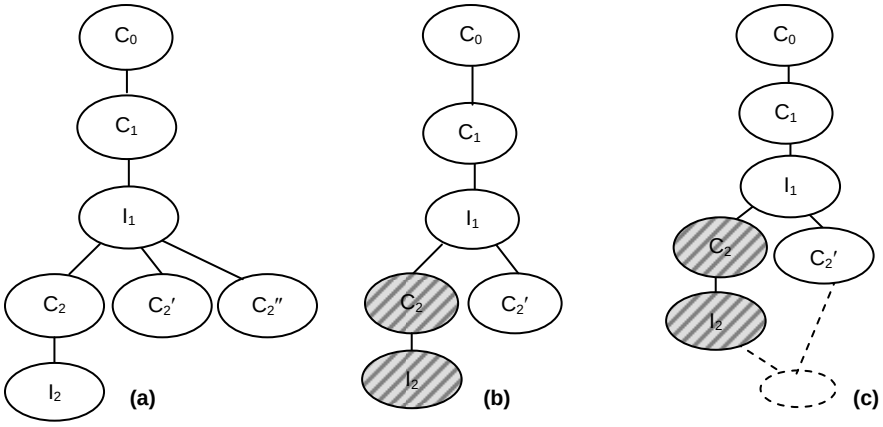

Figure 4. (a) A composition, (b) An execution of the composition, (c) A closed c-tree for the execution-tree

C_0, C_1, I_1 and C_2' with s-terminations and C_1 and I_2 with f-terminations.) Otherwise, that is, if only the activities from the root to some non-leaf node have been executed (for example, only C_0, C_1 and I_1) and/or the executions of some activities are not complete (C_2' is still being executed), then it is an *incomplete* execution of **C**. If *E* is a complete (incomplete) execution and each activity in *E* has s-terminated, then *E* is a *complete (incomplete) s-termination* of **C**. A complete s-termination is usually called simply as an *s-termination* of **C**. An *f-termination* of **C** is either a complete or incomplete execution in which executions of some activities have f-terminated.

- In each s-termination *C*, at each non-leaf node a_i, the selection of the s-terminated child of a_i satisfies the ce-predicate currently specified for $\mathbf{a_i}$ in **C**.
- Both the st-predicate and the ce-predicate at each node $\mathbf{a_i}$ may be changing as the execution of subsequent activities of **C** proceeds.
- Partial execution of **C** will be represented by a path from the root a_1 to some node a_i in the tree, and will be denoted (a_1, ..., a_i), and also as $C_{[1,i]}$. Here, the part that is yet to be executed to get a complete termination of **C** is the subcomposition of **C** from $\mathbf{a_i}$, called the *suffix* of **C** from $\mathbf{a_i}$, denoted $\mathbf{C_{[i]}}$. The subcomposition will contain the subtree of **C** rooted at $\mathbf{a_i}$, with the st-predicate and ce-predicate of $\mathbf{a_i}$ adjusted according to the execution $C_{[1,i]}$, and the st-predicate and ce-predicate of all other nodes in the subtree being the same as in **C**.

C. Transactional Properties

We first define transactional properties for basic activities.

- An execution a_i is said to be compensatable if there is a re-execution that will yield the null termination. It is re-triable if there is a re-execution that will yield a s-termination.

- An activity a_i is atomic if every execution of a_i guarantees either a complete s-termination or the null termination.

- Each activity a_i in *C* may first be weakly committed, and then strongly committed some time after its s-termination.
- Once a_i is weakly committed, as stated earlier, it cannot be compensated, and once it is strongly committed, it cannot be retried.

-The activities in C are (both weakly and strongly) committed in sequence. That is, when a_i is weakly committed, *all* activities that precede a_i in C and have not yet been weakly committed are also weakly committed. Similarly, strong commitments of the executions are also in sequence

We now state the transactional properties for the composition.

-Composition **C** assumes that each of its activities a_i is executed atomically. Thus an incomplete f-termination of a_i is assumed to be compensatable, to get an effective null execution**.**

- The execution of the entire composition **C** is intended to be atomic in **U**. (We elaborate this later.) That is, an execution of **C** should eventually yield a complete s-termination or the null termination.
- Consider an execution E of **C**.
  - If E is an incomplete s-termination, then forward-recovery is carried out by executing the suffix of E in **C** or a different acceptable sub-composition, to get a complete s-termination.
  - If E is either incomplete or complete f-termination, then the executions of some activities may have to be adjusted (partial backward-recovery) to get an incomplete s-termination, and a forward-recovery is carried out.
  - To get the null termination, E has to be compensated. This is the full backward-recovery.

D. Implementation Issues

(a) Point of Commitment

The execution of an activity a_i can be weakly committed any time, and then, after an s-termination, can be strongly committed any time. Weak commitment immediately after the s-termination gives pivotal property in the traditional sense. Waiting until the end of the execution of the entire composite activity will give the compensatability and retriability options until the very end. The longer the commitment is delayed, the more

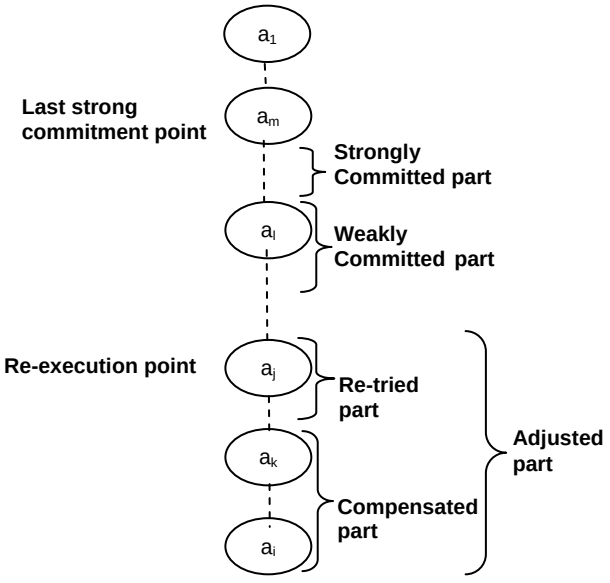

Figure 5. Partial backward-recovery in the Path model

flexibility we have for adjustment on execution of the subsequent activities. However, commitment of some subsequent activities may force the commitment of a_i.

(b) Adaptivity

As mentioned earlier, the ce-predicate will keep changing as the execution proceeds. (This is illustrated below, in Example 4.) Also, additional execution paths can be added, as descendents of a node, in the middle of the execution of the composite activity. Some execution paths may be deleted too. Thus, the composition could be adaptive and dynamic.

(c) Partial Backward-Recovery

Typically, the recovery starts with re-execution of a_j, for some $j \leq i$, where a_i is the latest activity that has been or being executed. If a_j has to be compensated, then all activities in the execution following a_j are also compensated, and a different child of a_{j-1} is chosen with possibly an updated ce-predicate at a_{j-1}. If a_j is retried, then, after retrying, a_{j+1} may need to be compensated or retried. Continuing this way, we will find that for some k, $k \geq j$, the activities in the sequence $(a_j, \ldots, a_{k-1})$ are retried and those in $(a_k, \ldots, a_i)$ are compensated. This is illustrated in the bottom half of Figure 5. (The top half is explained later.) The following example illustrates backward-recovery.

Example 4: In the composition of Figure 4(a), 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 adjustment and inspected, and the options C_2' and C_2'' explored. This would amount to rolling back I_2 and C_2, and re-executing C_1 and I_1, each with adjusted st-predicate. Here the adjusted ce-predicate for I_1' will have only C_2' and C_2'' options. Suppose C_2' is tried and the execution was successful. Then the execution-tree will contain all the nodes except C_2'', with C_2 and I_2 as f-terminations. This is shown in Figure 4(b). Here, nodes for the f-terminated activities are shaded.

In the above argument, the first activity a_{k+1} that needs to be compensated is determined after re-executing its preceding activity a_k. It is quite possible, in some cases, that a_{k+1} is determined even before re-executing its predecessors. It is also possible that for some of the activities in $(a_{j+1}, \ldots, a_k)$, their previous executions are still valid, that is, no re-executions are necessary. We simply take this as requiring "trivial" re-executions

In Figure 5, we note that if m is the largest index such that a_m is strongly committed, then $j > m$, and if n is the largest index such that a_n is weakly committed, then $k+1 > n$. This follows since, by the definitions of strong and weak commitments, executions of activities up to a_m cannot be retried and those up to a_n cannot be compensated. In the figure, a_n is not shown. It will be between a_m and a_{k+1}.

(d) Dependencies

Several dependencies are possible between execution states of different activities.

I. In general, any of the compensation, weak commit and strong commit actions on one activity may require any of these three actions for another activity. Such dependencies are similar to the abort and commit dependencies for database transactions given by Chrysanthis and Ramamrithm in [8]. They are given in Table 1. The '√' entries indicate the possibilities of the corresponding dependencies, and the '×' entries indicate the impossibility.

Table 1. Dependency-Table			
	a_j		
a_i	**Compensate**	**Weak Commit**	**Strong Commit**
Compensate	√	√	√
Weak Commit	×	√	√
Strong Commit	×	×	√

The relative positions of the nodes a_i and a_j are as in Figure 5, that is, a_i is a descendent of a_j. Each '√' entry in the table describes that the specified action in the execution of a_i requires the specified action in the execution of a_j, and also the dependencies where the roles of a_i and a_j are reversed. Recall that the s- or f-termination status of an execution may be known only at a later time. Hence, with respect to Figure 5, it is possible that the f-termination of a_j is known only after a_i is executed. Thus, it makes sense to talk about how the actions on a node affect the executions of its descendents. Note also the following.

- We assume that both weak and strong commitments are in top-down order. Therefore, if a_i is weakly committed, then a_j must be weakly committed too if it has not been done already. The same applies to strong commitment.
- If a_j is compensated, then a_i must be compensated too.

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

1. In our formalism, a change in the st-predicate of an activity may change the status of its earlier execution from s- to f-termination and hence warrant either a re-execution to get a new s-termination or compensation. That is, a change in the st-predicate value can account for both retrying and compensation. Therefore, we can define dependencies of the form:

- An f-termination of an activity changes the st-predicate of another activity and, in fact, of several activities.

2. Secondly, recall that the st-predicate is a Boolean expression of property-predicates. Then an f-termination means that some of these predicates are not satisfied. Depending on the property-predicates that are not satisfied, several f-terminations are possible. We allow for each of these f-terminations to change the st-predicates of other activities possibly differently. Therefore, we can expand the dependencies as follows.

- Each different type of f-termination of an activity changes the st-predicates of a set of activities in a specific way.

3. Dependencies involving s-terminations are also possible. We have seen that different s-terminations are possible. Each can affect other activities differently.

Therefore, a general form of dependencies is:

- A specific (s- or f-) termination of an execution changes the st-predicates of a set of activities in a specific way.

Note that this takes care of another case also: An execution of an activity a_k may be an f-termination (with respect to st-predicate prescribed for that activity) but, for some

reasons, we need to keep that execution. Then, the only way could be changing the st-predicates of some other activities which in turn change the st-predicate of a_k and make the current execution a s-termination.

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 a_i change the st-predicates of some other activities.

(The top half of Figure 5 shows the weak and strong commits triggered by the compensation or re-execution of the activities in $(a_j, \ldots, a_i)$.)

The execution of an activity a_i can be weakly committed any time, and then, after an s-termination, can be strongly committed any time. Weak commitment immediately after the s-termination gives pivotal property in the traditional sense. Waiting until the end of the execution of the entire composite activity will give the compensatability and/or re-executability options until the very end. The longer the commitment is delayed, the more flexibility we have for adjustment on execution of the subsequent activities. However, as we have seen above, executions and commitments of some subsequent activities may also force the commitment of a_i.

IV. Dependencies constraining the beginning of an execution of an activity can also be defined. For example, for activities a_j and descendent a_i possible dependencies are: a_i cannot begin execution until a_j (i) s-terminates, (ii) weak-commits, or (iii) strong-commits. Note that our composition model assumes that the execution of a_i cannot begin until the execution of a_j begins.

We end this sub-section with an example that illustrates some dependencies.

Procurement Example

This example is drawn from the contract for building a house explained in Section 1, that concerns with procurement of a set of windows for the 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 directed path containing nodes for each of the activities in the given order, as shown in Figure 6.

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, and, 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.

We describe several scenarios giving rise to different transactional properties.

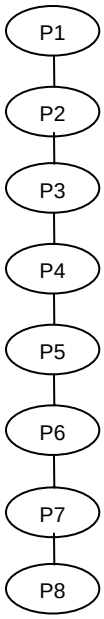

Figure 6. Procurement Example

1) Suppose that once the seller decides to accept the order, the order cannot be cancelled by the buyer or the seller, but modifications to the order are allowed, for example, delivery date changed, quantity increased, etc. If only the modifications that do not result in the non-fulfillment and hence cancellation of the order are allowed, then when the seller accepts the order, both P1 and P2 can be weakly committed. (On the other hand, if there is a possibility of the order getting cancelled, weak commitment has to be postponed. We do not consider this case any further in the following.)
2) There may be a dependency stating that the order can be sent to the manufacturing plant only after its acceptance by the seller, that is, the execution of P3 can begin only after P2 is weakly committed.
3) The plant may find that the goods cannot be manufactured according to the specifications, that is, P4 fails. Then the buyer may be requested to modify the order. For example, if the failure is due to inability to produce the required quantity by the due date, then the modification could be extension of the due date or reduction of the quantity or both. (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.
4) If the buyer finds that the goods do not meet the type specifications, that is, P7 fails, then, P4 has to be re-executed. In addition, P5 and P6 have to be re-executed. (This situation may arise also when the plant realizes some defects in the goods and "recalls" them after they were shipped.) Here, the re-executions may consist of the buyer shipping back the already received goods to the plant and the plant shipping the 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, perhaps by the same shipping agent (in which case the re-execution of P5 is trivial).
5) 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 re-executed. In the latter case, it follows that P6 will also be re-executed

4.2. Tree Model

We now present an extension, called the *tree model.* Here, we consider compositions that allow for more than one child to be executed at non-leaf nodes. Therefore, the execution yields a tree, instead of just a path, as a composite activity. The features of this model are essentially the same as in the path model. The difference is only in the complexity of the details. We outline the details in the following.

A. Composition

Here also, a composition **C** is a tree and it is a part of a higher level composition **U**. A st-predicate is associated with **C**. A st-predicate and a ce-predicate are associated with each node. These will be derived from the st-predicate of **C**. The ce-predicate is null for all leaves of **C**. The ce-predicate at non-leaf nodes may be sophisticated.

- More than one child may be required to be executed.
- In general, several sets of children may be specified with the requirement that one of those sets be executed.
- These sets may be prioritized in an arbitrary way.
- Execution of children within a set may also be prioritized in an arbitrary way.

B. Execution

An execution *E* of **C** yields a composite activity *C,* which is a sub-tree of **C,** namely, an *execution-tree*, such that

- It includes the root and some descendents;
- Some nodes are (fully compensated) f-terminations; If a node is an f-termination, then all descendents of that node in the execution tree are also f-terminations; and
- The execution of each s-terminated node satisfies the st-predicate prescribed for that node, and the non-f-terminated children of each non-leaf node of the sub-tree satisfy (fully or partially) the ce-predicate specified in C for that node.

An s-termination of **C** is an execution of **C** such that the non-f-terminated nodes yield a sub-tree of C that contains (i) the root, (ii) some leaves of **C** and (iii) all nodes and edges in the paths from the root to those leaves.

A partial execution *E* of **C** will be represented by a sub-tree of **C** consisting of all the nodes of **C** that have been executed so far and the edges between them. The suffix of the execution *E* can be defined similar to that in the path model. It will consist of sub-trees of **C** rooted at some of the leaves of the execution tree, with st- and ce-predicates properly adjusted.

C. Transactional Properties

The following definitions refine those given for the path model.

- We say that the execution a_i is *locally compensatable* if the execution can be undone to get the null termination. We also define another notion: a_i is *compensatable relative to C* if either a_i is locally compensatable or it can be compensated by executing a compensating activity within **C**.
- Similarly, an execution a_i is *locally retriable* if there is a re-execution that will yield an s-termination. And, a_i is *retriable relative to C* if either a_i is locally retriable or additional activities can be executed in *C* to get the effects of an s-termination of a_i.
- An execution a_i is *locally weakly committed* if it is locally non-compensatable (but locally retriable) and *weakly committed relative to C* if it is non-

compensatable relative to *C* (but retriable relative to *C*). The strong commit properties are similar.

- We define atomicity of a basic activity also in two ways: a_i is *locally atomic* if every execution guarantees either a complete s-termination or the null termination; and it is *atomic relative to C* if either it is locally atomic or (i) any non-null f-termination can be compensated by executing a compensating activity in *C* and (ii) any incomplete s-termination can be extended to a complete s-termination by executing additional activities in *C*. Composition **C** expects that each of its activities a_i is atomic relative to *C*.
- Again, the execution of the entire composition **C** is intended to be atomic in **U**. That is, an execution of **C** should yield a complete s-termination or the null termination. Therefore, if an s-termination of an activity a_i is not possible in some execution, then (that execution of a_i is compensated and) execution of a different set of children satisfying the ce-predicate of its parent is tried. If unsuccessful, then a different s-termination of the parent is tried. If not, similar adjustments at the grand-parent level are tried, and so on. Thus, either a complete backward recovery yielding the null termination or a partial backward recovery followed by forward execution to get an s-termination of *C* is carried out. Forward-recovery of *E* will consist of execution of the suffix of *E*. Partial backward-recovery of *E* will again consist of retrying the executions of some of the activities of the execution-tree, and compensating some others. This is illustrated in Figure 7.

D. Implementation Issues

All the issues discussed in the path model section are applicable here also. In the general case, where the execution-tree is a tree (see Fig. 7), the dependencies and the partial rollback are similar to the path case. The difference is only in the complexity of the details. All the dependencies discussed so far are applicable in the general case also, both for vertically (that is, ancestrally) and horizontally related activities. In addition, for horizontally related activities a_i and a_j, all combinations in the Dependency-Table 1 are possible, that is, all entries will be '√'. Dependencies that involve ce-predicates are also possible. A general statement would be:

- ❖ A specific (s- or f-) termination, compensate, weak and strong commit actions of an activity changes the ce-predicates of some other activities.

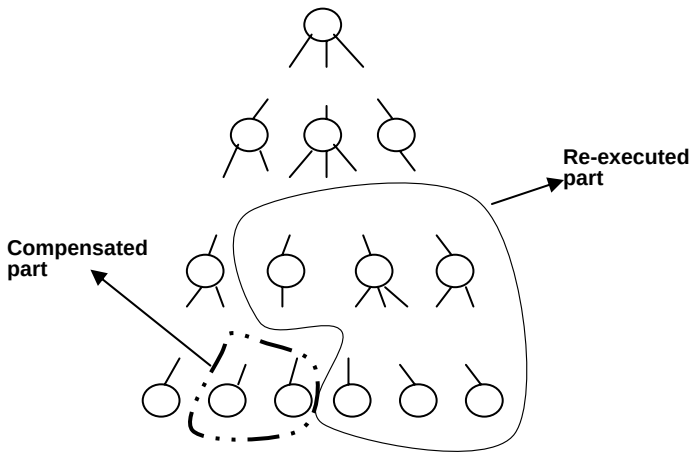

Figure 7 Partial backward-recovery in the Tree-model

We discuss some additional issues in the following.

(a) st- and ce-predicates

We have associated an st-predicate and a ce-predicate with each activity in our model. They are activity-dependent. We can expect that they can be expressed more precisely for some activities than for some others. In fact, for some activities, what constitutes s-termination may not be known until after an execution of that activity, and even after the execution of many subsequent activities. We note also that the st-predicate of a composite activity determines the st-predicate and the ce-predicate of its constituent activities. Hence, specification of the st- and ce-predicates is crucial. This will be the role of the (activity and) workflow semantics.

Whereas the semantic specification of ce-predicate would be application-dependent, syntactic specification may be made more precise, with an appropriate language. We can expect that such a language would have constructs for specifying priorities and Boolean connectives. An example is booking an all (flight-hotel-food) inclusive package, and if it is not available then booking flights and three-star hotels separately, for a vacation.

The ce-predicate allows specifying preferences in the selection of the children activities to be executed. Preferences may exist for s-terminations too. This may depend on functional as well as non-functional aspects of the execution. Such preferences can be incorporated in the model easily.

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.

(b) Closure of Composite Activities

A composite activity *C* also can be closed in three different states depicted in Figure 2, namely, null termination, (incomplete or complete) non-null f-termination, and (complete) strongly committed s-termination. The null execution might be the result of executing a compensating activity. Therefore, in any of these terminations of *C,* the constituent activities of *C* might be closed in any of the three terminations. Now, *C* may be closed either before or after some or all of the constituent activities of *C* are closed. An example of the former would be not waiting for the closure, or even termination, of some activities that compensate some other activities in the original execution of *C,* that are guaranteed to succeed

Procurement example revisited: In the example illustrated in the last sub-section, suppose the seller splits the order into two parts and assigns them to two plants Plant-A and Plant-B. Then the node P3 will have two children and its ce-predicate will contain the details of the individual orders. Corresponding to P4, P5 and P6, we will have P4-A, P5-A and P6-A for Plant-A, and P4-B, P5-B and P6-B for Plant-B. This is shown in Figure 8. We describe a few scenarios and the resulting dependencies.

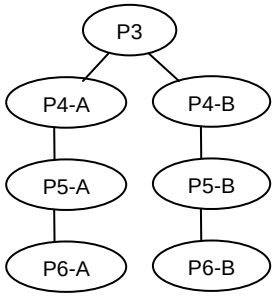

Figure 8. Procurement example with two plants

1) The seller may decide that shipping should not start

until all the goods in the order have been manufactured. The gives rise to the dependencies: begin P5-A and P5-B only after both P4-A and P4-B s-terminate.
2) P5-A fails, that is, Plant-A is unable to find a shipping agent. Then, the shipping agent of Plant-B may be asked to ship the goods of Plant-A also. This may involve changing the st-predicate if the execution of P6-B has not been done or re-execution of P6-B otherwise.
3) The buyer is not satisfied with the goods manufactured in Plant-A, that is, P7 fails. Then, the seller might settle for the buyer returning those goods and Plant-B manufacture those goods and send to the buyer. This involves changing the ce-predicate at P3, compensation of P4-A, P5-A and P6-A, and re-execution of P4-B, P5-B and P6-B.

4.3. Multi-Level Model

So far, we have dealt with compositions at a single level, in fact, the bottom-most level where all activities are basic activities. Now we extend the model by allowing basic or composite activities in the compositions. This gives us a multi-level, hierarchical, composition model. The highest level activity is the contract-activity. In the previous sections, a composition **C** is described as a tree. An execution of **C** yields a composite activity *C*, which is a path graph in the path model and a tree in the tree model. We call (both of) them a *composite activity tree,* or *c-tree* in short.

An outline of the multi-level model is the following.

- **Composition**

A composition **C** is a tree as in the tree model. Nodes in the tree are (sub)compositions of basic or composite activities. Compositions of composite activities are, again, trees as in the tree model. Thus **C** is a "nested" tree. An st-predicate is associated with **C**.

- **Execution**

Execution of each subcomposition of **C** yields a c-tree. (For a basic activity, the c-tree will have just one node.) To put these trees together, we use the following notation. A 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.* Figure 4(c) shows a closed c-tree for the execution-tree in Figure 4(b). Figure 9 illustrates the two-level composition for the Procurement example.

In the execution of a multi-level composition **C**, at the top level we get a closed c-tree with nodes corresponding to the executions of activities in **C**. Each of the activities will again yield a closed c-tree. Thus, the graph can be expanded until all the nodes correspond to basic activities.

Partial execution is considered as in the tree model, level by level, in nested fashion.

- **Transactional Properties**

At each individual level, for each node, the transactional properties discussed with the tree model are applicable. After the recovery of one node, the recovery efforts at the parent level execution will continue.

We have already discussed s-terminations and f-terminations of composite activities. We can define compensatability, retriability and commit properties as well as atomicity for composite activities as we did for basic activities, namely, both *locally* as well as *relative to the parent level composite activity U*. For example, *C* is *locally compensatable*

if the null effect can be obtained by simply modifying the composition **C** and executing, and is *compensatable relative to U* if it is compensatable either locally or by executing a compensating activity, say **C′**, within **U.** In the latter case, **C′** will also be specified as a tree with suitable st-predicate. (For example, if the original execution is building a garden shed in the backyard, the compensation might be the demolition of that shed.)

We can also extend these definitions across multiple levels. For example, in the above case where *C* is compensatable relative to *U,* we say that a_i is also compensatable relative to *U* even if a_i is not compensatable relative to *C.* By this, we mean that the effects of a_i can be compensated either by compensation of *C* by *C′* or by a compensating activity a_i', both in *U.* The definitions for retriability are analogous. Thus, in general, re-execution of a composite activity would require adjusting the composition of that activity in terms of adding and/or deleting some nodes and adjusting the st- and ce-predicate of the nodes. This can also be thought of as coming up with a new composition for that activity, mapping the previous execution on the new composition, identifying the s-terminated part, and doing a backward- and/or forward-recovery. The re-execution and adjustments of the st- and ce-predicates will then be top-down.

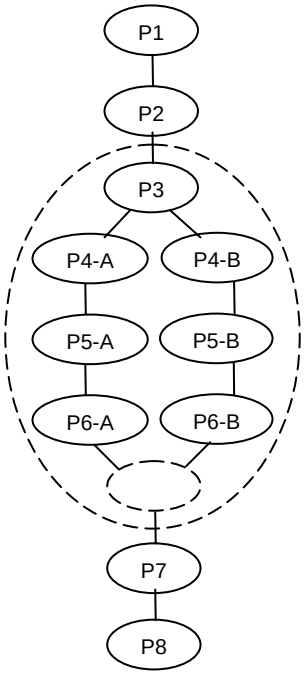

Figure 9. Two-level composition for the Procurement example

Example 5: In Example 2, suppose the addresses are printed and the stamp glued, and we find later that the To address is incorrect. If the affixed stamp cannot be removed, the activity a_2 is non-compensatable, but only relative to *C*. The activity *C* itself may be compensatable relative to *U*, amounting to just tearing up the envelope and bearing the loss of the stamp. Then, though a_2 itself is not compensated the composite activity containing a_2 is compensated.

Similarly, the commitment properties at the two levels are also independent of each other. We give two examples. (1) Activity a_i could be strongly committed, meaning that it cannot be compensated or re-executed in *C*, but *C* itself may be weakly committed

relative to U, meaning that it may be re-executed perhaps with additional activities. C could be weakly committed even if some activities of C are not executed yet, if retrying of C can be carried out by compensating the current execution completely and re-executing it to get an s-termination. (2) An example of a_i being weakly committed and C being strongly committed is that of fixing (perhaps in the warranty period) a broken pipe after the construction of the house is finished and the builder paid fully. Thus our model allows, as mentioned in Section 1, re-executing even a "committed" activity, by dealing with commitment in multiple levels.

D. Implementation Issues

All the issues discussed for the single level (bottom level) composition are applicable here also within each level, and also across levels. For example, suppose as before that a_i is an activity of C which is a composite activity of U, and C' is another composite activity of U. We may have a dependency of the type: if a_i is strongly committed then C' has to be compensated.

Some of the activities (usually high level ones) will correspond to parts of the contract or subcontracts. As noted earlier, at the highest level, the composition is for the entire contract-activity. On closure of such activities, the corresponding contracts themselves might be closed. Closure of a contract intuitively refers to expiring the "life" or validity of the contract. For example, a contract for building a house may close after the warranty period during which the builder is responsible for repairs. A sub-contract for maintaining an air-conditioning system installed in that house may close at a different time. The transactional properties in our model can be used to refine the conditions for closure of the contracts.

5. Monitoring Payments

In this section, we address the vital issue of payments in e-contracts. Payments are meant for the execution of the activities in the e-contract. Hence, we should be able to ascertain that the activities have been executed (or compensated) satisfactorily to deserve payment. We are concerned with three critical aspects that dictate the payments for an activity: the cost of execution of the activity; the amount of payment for the execution; and the time of payment for that activity. All these require a good understanding of the execution states of the activities and hence the e-contract.

5.1. Cost and payment

Below we discuss different ways of assigning cost and the amount of payment for an execution. Let C be a composite activity consisting of basic activities a_1, a_2, etc. There are two aspects – cost of execution of an activity a_i (i) *for* a_i and (ii) *for* C, that is, the cost charged to C and hence to be paid by (the service executing) C to (the service executing) a_i. We look at both of them. We use $cost(a_i)$ and $payment(a_i)$ to denote (i) and (ii), respectively.

Cost:

- A cost may be associated with each execution related to a_i. The total cost $cost(a_i)$ will depend on the number of executions related to a_i.

- Different s-terminations are possible. They may have different costs. (For example, fully refundable flight tickets normally cost more than non-refundable tickets.) We will not concern about how the costs are arrived at.
- A non-executed null termination will cost nothing. If an activity a_i has been executed and then compensated, even if the resulting execution is effectively null, a cost may have to be associated.
- With non-null f-terminations also, a cost may be associated.
- Each re-execution may incur additional cost. Therefore, as the number of re-executions (as depicted in Figure 3) increases, the cost of the activity $cost(a_i)$ will keep going higher.
- Therefore, the final cost of execution, $cost(a_i)$, will depend on the number of (re-executions and hence) terminations, and will be known to a_i only when its execution is strongly committed.

Payment:
- Payment for a_i, namely, $payment(a_i)$, may not directly depend on the number of executions related to a_i.
- The cost of an execution resulting in a f-termination and the cost of compensating that execution may not be charged for the payment.
- When several re-executions are done, the costs for some of them may not be charged.
- The above considerations apply when the payment amount is determined after the execution of the activity. However, depending on which costs are not charged to C, $payment(a_i)$ may be known earlier. For example, if the charge is zero for a compensated a_i, it can be known even before the compensation is complete, and if re-execution costs are not charged to C, then $payment(a_i)$ can be known on weak commitment of a_i.
- On the other hand, $payment(a_i)$ could be fixed even before the execution of a_i starts, for example, when the parties are entering into contracts. Then the amount could be based on the number of anticipated re-executions and the prospects of arriving at a satisfactory s-termination eventually.

The cost and payment for a composite activity will depend on the costs and payments for the individual activities in the composition that are executed.

We now illustrate some scenarios in the Procurement example of Section 4.

- When the goods are not delivered on time (P6), the buyer can insist on canceling the order. Then, a cost is incurred in the initial delivery of the goods as well as in the return of the goods as part of canceling the order, though the effective execution of that activity is null.
- Consider another related clause: “If the goods are not confirming as per the contract, the buyer may require the seller to remedy the lack of conformity by repair.” Then further costs are involved in returning the goods, repairing them and sending them back to the buyer.

There are two aspects for payment(s) for an activity also – *enabling* payments and *making* payments. Payment options include the following:

- For each activity, either a single payment or multiple (partial) payments may be enabled at various terminations depicted in Figure 3. Similarly, payments can be made just once or in several installments. The number of installments need not correlate with the number of enabling points.
- A payment can be (partly or fully) refundable or non-refundable. In the former case, we need to calculate refund with respect to payments made previously (for example, when some activity that has been prepaid has not been executed). In the latter case, making payments may be delayed until a good estimate of the cost of execution is obtained.
- As stated earlier, the actual cost of execution may be known only after strongly committed s-termination or f-termination. Then, any payments done in other states may have to be adjusted at the end. We will not consider the details (such as the time and the amount) of the adjustments in this paper.

A payment monitoring system should keep track of the state of termination, *payment-enabled* and *payment-made* points and the amounts, for each activity.

5.2. Payment for Basic Activities

We first consider the bottom-most level, where each composite activity is composed of basic activities; the general model for activities at multiple levels is considered in the next sub-section. The same composition model described in section 4 is applicable for handling payments.

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

- For each activity, payment(s) may be enabled and made in any of the terminations of execution of that activity, as discussed earlier (see figure 3), and also in the states of weak or strong commits relative to *C*. In addition, payment for a_i may be enabled either when a_i 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**.
- If an execution a_i is compensated by execution a_i' of a compensating activity, then both a_i and a_i' will appear in the execution-tree, and costs may be attributed to them individually.
- Similarly, if re-trying of a_i is done by executing additional activities, their executions will also be in the execution-tree and costs can be assigned to them.
- Enabling and making payments for different activities can occur at different times.
- Dependencies may exist between enabling/making payments of different activities.
- Dependencies may also exist between enabling/making payments for one activity and starting the execution of (and similarly, compensating, weakly committing and strongly committing) another activity, and vice versa.
- At any stage, the activities whose payments have been enabled and those whose payments have been made are kept track of with a *payment-enabled-tree* and a *payment-made-tree*, respectively.

We 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.

Example 6: For the case discussed in the Example 4, a payment-made-tree could have all the nodes except I_2. This is shown in Figure 10. Comparing with the execution-tree described in Figure 4(b), payment for I_2 has not been done yet, and 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 the non-executed activity has to be adjusted later on.

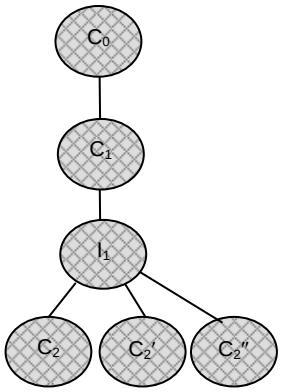

Figure 10. A payment-made-tree for the composition

The cost of an execution *E* of a composite activity *C* is simply the sum of the payments for the executions of its constituent activities.

Below we present a small example to illustrate the execution of activities and terms of payments. In the house-building contract, consider a sub-task involving construction of a wall and painting it. The activities are shown in Table 2. The work begins with the 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). Once all the materials are gathered, a 20cms thick wall is constructed as per the building specifications. After this process, 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

Table 2. 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

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) and 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. Eventually, when the wall is constructed to the satisfaction, it is painted.

The re-execution and compensation details are given in the table. Re-executions carried out after weak commits are stated as retrys. Weak and strong commit points for the various activities are also given. 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.

The table states the terms for three payments: when they are due and 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.

Table 3. Costs for execution of the activities described in Table 2

Payments	**Activities Related**	**Cost Incurred (as per Column 3, Table 2)**
Payment-1	Activity 1 and Activity 2	C1 + C2 + RE1a + RE2
Payment-2	Activity 3 and Activity 4 Re-executions of 1 to 4 if any.	C3 + C4+ RE3a + CS3b + RE1a + RE2 + RE3c + C4
Payment-3	Activity 5 and Activity 6	C5 + C6 + RE5a + RE 5b + C6 + RE5b + C6

Table 3 shows the costs while executing the activities as given in Table 2. The cost of the first execution, compensation and re-execution are denoted as C, CS and RE respectively, followed by the activity number specified in Table 2. Note that, in a normal straightforward situation, the expected cost of the composite activity for construction and painting of the wall is C1 + C2 + C3 + C4 + C5 + C6. However, due to multiple re-executions and compensation, the total cost may become higher than the expected cost.

5.3. Multi-level Composition

For computing the cost as well as keeping track of the payments, a straight-forward approach is to use the (multi-level) execution-tree and similarly the corresponding payment trees. All these will be updated, as the execution proceeds and payments are enabled and made. One way of computing the cost of a composite activity is considering the single level execution tree of that activity. For example, in a composition **U** consisting of a composite activity **C** which again consists of basic activities a_is, *payment*(a_i) of each a_i (which is the amount charged to *C*) is added to get *cost*(*C*). However, *payment*(*C*) may be different, and this is the amount used, for each constituent activity of *U,* to compute *cost*(*U*). Thus the costs can be computed recursively, in bottom-up fashion.

For keeping track of the payments also, several sub-trees (of the fully nested payment trees described above) can be used depending on the level of detail that we are interested in. For example, considering the composite activity **C**, if at the level of **U**, only the lump-

sum payment for **C** is of interest, then there is no need to expand the node corresponding to **C** to the sub-tree representing the execution of the activities of **C**. (This latter sub-tree will be used at the level of **C**.) This will be appropriate, for example, when **C** is a sub-contracted activity, executed and managed by a different party.

The extended definitions of transactional properties allow enabling and making payments in sophisticated ways. For example, payment for a_i can be enabled only when it is weakly committed relative to *U*. This policy might be appropriate when payment authorizations come from *U* and not from *C*.

The various dependencies can be defined between activities at different levels too. For example, payment for a_i may be enabled after payment for *U* is enabled and irrespective of whether payment for *C* is enabled. Another example is starting the execution of some activity **D** in **U** only after payment is made for a_i.

5.4. Discussion

We have expressed that costs are determined by the executions. It is also possible that costs and payments influence the execution. Examples are:

- We associated a cost for each retry. 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.
- Payment 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 can be expected after payment, then strong commitment can be combined with the payment.

Activities in e-contract may be executed autonomously. Then, 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.

In the literature, the inter-dependency among contract satisfaction, activity execution and payments has not been explicitly modeled. The utility of such modeling in deploying and managing the commitment and payment aspects of e-contract is immense. Some of the open issues in this problem domain 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. It is expected that, in future, e-contract management systems will seamlessly process contracts and monitor their completion and payment aspects.

In contracts, payment terms are arrived at based on negotiations. Normally, we can expect that all payments will be made before the closure of the contract. However, there are exceptions some of which may be very sophisticated. An example is a contract for the construction of a flyover in which (a part of) the payment is through toll gate collection for a few years after the construction is finished.

6. Conclusions

In this paper, we have presented a framework for e-contract commitment by considering transactional properties for executions of activities of the e-contract. To the

best of our knowledge, this has not been done in the literature so far. Accommodating the transactional properties can improve an e-contract design and, in turn, help in the enactment of the underlying contract. Some important aspects are the following.

i. 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.
ii. Closure of the contract can be tied to closure of the activities and commitments. Features such as "the life of a contract may extend far beyond the termination of execution of the activities" are accommodated fairly easily.
iii. Terms of payments for the activities (including the contract-activity) can be related to the execution states of the activities.

The transactional properties described in this work will be useful in other applications also, irrespective of whether the activities are electronic, non-electronic or both.

An e-contract system must ensure the progress of activities and their termination. Since e-contracts consist of multiple activities executed with several inter-dependencies, any failure could have cascading effects on other executed or executing activities. In this work, we have also brought out these dependencies explicitly and facilitated solutions that can be incorporated within an e-contract system. This study will be helpful in monitoring behavioral conditions stated in an e-contract during its execution.

References

[1] Alonso, G., Agrawal, D., Abbadi, A.E., Kamath, M., Gunthor, R., and Mohan, C., Advanced transaction models in workflow contexts, In Proc. of the 12th International conference on Data Engineering (ICDE '96), pp. 574-581,1996.

[2] Chen Q. and Dayal U., A Transactional Nested Process Management System. Proc. 12th Intl. Conf. on Data Engineering,pp.566-573, 1996.

[3] Chiu, D. K. W., Karlapalem, K., Qing Li. and Kafeza, E.: Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment, Distributed and Parallel Databases, 12, 2/3, 193-216 (2002).

[4] Chiu, D. K. W., Cheung, S. C. and Till, S., A Three-Layer Architecture for E-contract Enforcement in an E-Service Environment, 36th Hawaii ICSS, 2003.

[5] Chrysanthis, P. K., and Ramamritham, K., ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. Proceedings of the ACM SIGMOD International Conference on Management of Data: 194-203, 1990.

[6] Chrysanthis, P. K. and Ramamritham K.: A Formalism for Extended Transaction Models, Proc. of the 17th Int. Conf. on Very Large Data Bases, 103–112 (1991).

[7] Desai N., Narendra N. C. and Singh M. P., Checking Correctness of Business Contracts via Commitments, Proc. of 7th International Conference on Autonomous Agents and Multiagent Systems (AA-MAS 2008), Estoril, Portugal, 2008.

[8] Dayal U., Meichun Hsu, Ladin R., A Transactional Model for Long-Running Activities. VLDB 1991: 113-122, 1991.

[9] Farrell A. D. H., Marek J. Sergot, Bartolini, C., Salle, M., Trastour, D. and Christodoulou, A., Using the Event Calculus for the Performance Monitoring of Service-Level Agreements for Utility Computing. First IEEE Int. Workshop on Electronic Contracting, 2004.

[10] Grefen, P. and Vonk, J.: A Taxonomy of Transactional Workflow Support, International Journal of Cooperative Information Systems, 15, 1 87-118 (2006).

[11] Grefen, P., Aberer, K., Hoffner, Y. and Ludwig, H., CrossFlow: Cross-Organizational workflow management in Dynamic virtual enterprises, Int. Journal of Computer Systems Science and Engineering, 15 (5) (2000) 277-290.

[12] Grefen, P., Vonk, J. and Apers, P., Global transaction support for workflow management systems: from formal specification to practical implementation. The VLDB Journal 10, pp.316-333, 2001.

[13] Grosof, B. and Poon, T., SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies and Process Descriptions, Proc. of the 12th WWW Conference (2003).

[14] Jain A.K., Aparicio IV M. and Singh M. P., Agents for Process Coherence in Virtual Enterprises, Communications of the ACM, 42(3), (1999) 62-69.

[15] Kafeza, E., Chiu, D. and Kafeza, I.: View-based Contracts in an E-service Cross-organizational Workflow Environment, In: Proc. of 2nd Int. Workshop on Technologies for E-service (2001).

[16] Koetsier, M., Grefen, P., and Vonk, J.: Contract Model, Technical Report, Cross-Organizational/Workflow, Crossflow ESPRITE/28635 (1999).

[17] Krishna P. R., Karlapalem K. and Dani A. R., From Contracts to E-contracts: Modeling and Enactment, Information Technology and Management, 6, (2005), 363-387.

[18] Papazoglou M. P., Web Services and Business Transactions, World Wide Web: Internet and Web Information Systems, 6 (2003), 49–91.

[19] Rouached, M., Perrin, O. and Godart, C., A contract-based approach for monitoring collaborative web services using commitments in the event calculus, WISE05, 426–434, 2005.

[20] Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation for Transactional Processes, ACM Transactions on Database Systems, 27, 63-116 (2002).

[21] Vidyasankar, K.: Atomicity of Global Transactions in Distributed Heterogeneous Database Systems, Proc. of the DEXA-91, 321-326 (1991).

[22] Vidyasankar, K. and Vossen, G.: A Multi-Level Model for Web Service Composition, In: Proc. of the 3rd IEEE International Conference on Web Services, San Diego, U.S.A., pp 462-469 (2004).

[23] Vidyasankar, K. and Vossen, G.: Multi-Level Modeling of Web Service Compositions with Transactional Properties, Technical Report, Memorial University, St. John's, Canada, (2007).

[24] Vidyasankar K., Radha Krishna P., and Kamalakar Karlapalem, A Multi-Level Model for Activity Commitments in E-contracts, CoopIS 2007, Portugal, LNCS 4803 Part 1, pp. 300-317, 20072007.

[25] Vidyasankar K., Radha Krishna P. and Kamalakar Karlapalem, Study of Execution Centric Payment Issues in E-contracts, 2008 IEEE International Conference on Services Computing

(SCC 2008) July 8-11, 2008, Honolulu, Hawaii, USA, IEEE Computer Society, Vol 2, pp. 135-142, 2008.

[26] Vidyasankar K., Radha Krishna P. and Kamalakar Karlapalem, Study of Dependencies in Executions of E-contract Activities, 13th East European Conference on Advances in Databases and Information Systems (ADBIS), LNCS 5739, pp. 301-313, 2009 2009.

[27] Wang, T., Grefen, P. and Vonk, J.: Abstract Transaction Construct: Building a Transaction Framework for Contract-driven, Service-oriented Business Processes, In: Proc. of the ICSOC- 2006, LNCS 4294, Springer, pp. 434-439 (2006).

[28] Xu, L.: A Multi-party Contract Model, ACM SIGecom Exchanges, 5, 1, 13-23 (2004).