Recovery Guarantees in Mobile Systems

Cris Pedregal Martin and Krithi Ramamritham
Computer Science Department
University of Massachusetts
Amherst, Mass. 01003–4610
cris, krithi@cs.umass.edu

Abstract

Mobile applications increasingly require transaction-like properties, particularly those of recovery. Because there is a lack of abstractions to decompose the machinery of recovery, realizing recovery is difficult and error-prone, especially in a novel context like mobile systems.

We introduce *recovery guarantees* to tackle this problem by characterizing the assurances relevant to recovery that a subsystem must give to another. They describe the *what* can be expected but not the *how* it is implemented for recovery. Guarantees are complemented by recovery protocols, which prescribe behaviors subsystems should follow in order to take advantage of the guarantees.

In this paper we use the notions of recovery guarantees and protocols to show the relationships, vis-à-vis recovery, between the components of a mobile system. Our analysis shows which components of recovery remain unchanged (from a conventional recovery design) and which respond to the particular needs of mobile systems. This sheds light not just on how to do recovery on mobile systems but also on the nature of recovery in general.

1 Introduction

Recovery is important because it supports desirable transactional functionality, which preserves the consistency of data in the face of failures. The particular case of recovery in mobile systems is interesting for several reasons. First, mobile systems are multi-node distributed systems and, although they exhibit partial failures, their services are expected to degrade gracefully, i.e., with a loss of functionality proportional to the severity of the failure. Second, the mobility of a host imposes restrictions (on bandwidth, power, and reliability of storage) that affect key aspects of recovery support. A mobile system's reliable/stable storage may be limited, or nonexistent, necessitating the mobile host's frequent communication with the base; however, the bandwidth of the link between a mobile and its base is not only much smaller than that between computing and storage (disk) subsystems, but also smaller than that between hosts on a fixed (local or even wide-area) network. In other words, a mobile host needs its fixed base host to support its recovery, but communicating with it is slow and expensive. Also, when a mobile host migrates it changes base hosts, thus the mobile relates (for its recovery support) not with an individual fixed host but with the network, which becomes burdened with abstracting the migration for the purposes of recovery. Thus the problem of recovery in mobile systems is challenging because of its distributed nature and because resources are quite limited.

Our goals

Our research efforts are aimed at bringing the benefits of abstraction and the methodologies abstraction enables to the characterization and crafting of recovery functionality in a broad range of applications and infrastructure.

This paper is an exercise, as part of those efforts, to apply a novel abstraction (*recovery guarantees*) to expose the relationships, vis-à-vis recovery, between the components of a mobile system. Our goal is to show how mobility affects the realization of recovery properties, and to demonstrate the usefulness of our approach to reason about and craft recovery.

We believe that exposing the components of recovery and their relationships will improve modeling and crafting of recovery. This will improve the understanding of recovery, leading to an increase of confidence in existing schemes (less obscurity) and making tradeoffs more apparent, thus allowing more informed choices of infrastructure and protocols. Also, it will make it easier to build recovery with both traditional and novel database applications.

Organization of this paper

The rest of the paper is organized as follows. Section 2 reviews recovery concepts and mentions previous work on what recovery is and how it can be implemented. This leads to the introduction of recovery guarantees and protocols, followed by with notation and illustrated by two simple examples. Section 3 discusses a mobile system and its recovery characteristics, introducing the necessary notation. This is followed by a discussion how recovery information must be propagated to support eager and lazy handoffs, and how repair is done. Section 4 discusses related work and Section 5 offers conclusions and suggests further research.

2 Recovery Concepts

It is well understood that in general recovery is about carefully and economically preserving information sufficient to bring a system to a consistent state in the face of (system or transaction) failures. This is a reflection of two facts. One, transformations of a system from one consistent state to another are not atomic, hence failures midway leave the system in an inconsistent state if not repaired. Two, system state (partially) resides in, and thus its transformations happen on, (volatile) storage whose contents are lost when failures occur, necessitating the transcription of crucial state information to persistent storage.

The transactional properties of Failure Atomicity (FA) and Durability are the high-level requirements that characterize recovery. FA, also known as all-or-nothing, means that either all the operation by a transaction are installed (for a committed transaction) or none is. Durability means that once a transaction has been declared committed, its effects must persist on the system, in spite of failure. In order to maintain Failure Atomicity and Durability, a system must ensure that there is always enough information available so that even if a failure occurs, the system will be able to guarantee the following two requirements:[1]

Uncommitted ⇒ Undoable. The effects of operations by any uncommited transaction on a subsystem can be undone, even after a failure.

Committed ⇒ Durable. The effects of operations by a committed transaction will persist on a subsystem, in spite of failures.

Notice that these conditions do not describe *how* the recovery process works: they simply state *invariants* that *normal processing* must observe for recovery to be possible in the event of a failure. That is, these conditions describe *what* recovery is. The fundamentals of recovery in these abstract terms are covered in the literature [1, 4]. Most literature describes how to implement recovery [8, 2, 4] in very low-level terms, making it difficult to re-apply the basic ideas to novel scenarios. In other words, recovery is either discussed in general terms or in very detailed ones, with little work in-between [5]. This paper is an attempt to reduce the gap between the high-level requirements and their low-level implementations.

Recovery Guarantees and Protocols

We propose a more detailed but still generally applicable way of talking about recovery by introducing the notion of guarantees and protocols. Recovery is possible because different subsystems fail in different manners: an understanding of those failures (or, in other terms, of the reliability a subsystem has to offer), coupled with disciplined communications between subsystems, yields an aggregate system which is reliable as a whole in spite of the unreliability of its parts. Specifically, we describe the expectation a subsystem has on the reliability of another using the concept of *guarantee*.

In order to characterize the recovery properties of a system, we need to: (i) identify the relevant subsystems and their relationships, (ii) describe what each subsystem expects from others (the guarantees), and (iii) specify how subsystems behave in order to create the conditions for the guarantees to be useful (the protocols). Naturally, we do this in the context of the requirements of recovery (e.g., FA + D or a variant) and the infrastructure of requirements (e.g., stable logging facilities). We will not go into the details of (i): we assume a subsystem is a whole host in a mobile system, but it will be apparent that the ideas expressed here may be used to decompose a subsystem into smaller subsystems to understand its realization of recovery. For example, a fixed host may rely on some disk storage, either local or across the network, to provide its guarantees. Figure 1 gives examples of the recovery concepts and their relationships. In the sequel we concentrate on the guarantees and protocols.

Recovery guarantees describe the assurances relevant to recovery that a subsystem (e.g., a fixed host in a mobile system) must give to other subsystems (e.g., a mobile host). In general, recovery guarantees formalize expectations of reliability between parts of a system and have a broader applicability. Notice that, at a given level of abstraction, a guarantee describes *what* a subsystem offers as a recovery assurance, but not *how* it goes about doing it; in turn, how the guarantee is supported may be described in terms of guarantees offered by lower levels of the system. The notation $pred \leadsto q$ (where $pred$ is a predictate and q is an operation) means:

if $pred$ holds,
then if invoked, q will be performed successfully.

Thus, a *recovery guarantee* is a promise that once $pred$ is true, the guaranteed operation q, *if invoked*, will succeed, in spite of any intervening failures.

Often *where* an operation is performed will be part of the guarantee specification. In general, the protocol is as follows: a (requester) subsystem A invokes an operation p of (recovery, i.e., guarantor) subsystem B; the success of p (satisfies a predicate which) signifies that B guarantees to A that a future invocation of q will succeed. Basically, with the information made available by p (and perhaps other information maintained by B), B will be in a position to apply q, if requested. It should be noted, however, that the subsystem on which p is performed and the subsystem guaranteeing q need not be the same!

Also, note that we do not talk about the "commitment" or "durability" of p. All that is stated above is that p is executed at B for B to guarantee that, when invoked, q will be executed. How B ensures that this guarantee will hold (say, by making p's results durable, etc.) is of interest only when one wants to "verify" the capabilities of subsystem B.

Recovery protocols are prescriptions of how subsystems must behave (i.e., issue their operations) in order to avail themselves of

[1] Described as the redo and undo rules by Bernstein et al. [1].

	requirement	guarantee	protocol
examples	Failure Atomicity Durability	$R(p) \in B \rightsquigarrow p$	$WUL(p) \rightarrow p$
what they are	system property, seen by user and applications	subsystem property, supported by recovery manager	execution property, supported by data manager
how implemented	guarantees + protocols	stable logging, recovery	logging operations

Figure 1: Recovery concepts and their relationships.

the benefits given by the guarantees. For example, the protocol of logging undo information to disk before updating enables the data manager subsystem to rely on the guarantee of the disk's persistence in case the update needs to be undone later.

For the purposes of this paper, we use protocols that satisfy certain precedence constraints. To this end, notation $p \rightarrow q$ (p, q operations) is used. It means:

if q has been performed,
then p must have happened before.

The rest of the notation follows.

Notation

We represent the behavior of the system with events in a partial order –the history of the system. The subscript denotes in which subsystem the operation that the event represents takes place. The meanings of $\rightsquigarrow$ (guarantees) and $\rightarrow$ (precedes) were defined in the preceding paragraphs. Symbols $=$ and $\Rightarrow$ have the usual meanings of identity and logical implication, respectively.

- $A, B, C, ...$ denote subsystems on which operations can be issued and/or applied. Example subsystems: magnetic disk; hosts in a network.

- p_A denotes a generic data operation which modifies the state of the subsystem A on which it is applied. Note that p may be invoked in a subsystem other than A, and applied in A. The inverse of p_A is written p_A^-.

- $R(p_A)$ denotes the information necessary to reproduce the effects of operation p_A (on subsystem A). $R(p_A)$ is a data item and may be stored and transmitted; when subsystem B possesses such data we write $R(p_A) \in B$.
  A way to obtain $R(p_A) \in B$ is to write a redo log record, which we denote this write operation $WRL_B(p_A)$.

- $U(p_A)$ denotes the information necessary to undo the effects of operation p_A (on subsystem A). $U(p_A)$ is a data item and may be stored and transmitted; when subsystem B possesses such data we write $U(p_A) \in B$.
  A way to obtain $U(p_A) \in B$ is to write an undo log record, which we denote this write operation $WUL_B(p_A)$. $U(p_A)$ may be used to effect p_A^-.

Example: Redo Log

Writing the redo log for p on B guarantees that it will be possible to apply p on A (in spite of failures):

G1 : $R(p_A) \in B \rightsquigarrow p_A$

The ability to offer the guarantee G1 requires that:

- B have means to store the log record for $R(p)$ that will survive failures. The guarantee acts as a requirement of persistence on B. (B may be a disk subsystem, or a network connected node whose failure mode is independent of A's.)

- The recovery system be able to use the log record as $R(p)$ to reproduce the effects of p if necessary.

Guarantee G1 is useful to ensure that it is always possible to effect operations issued by a committed transaction. To use the guarantee for that purpose, the system must follow protocol P1:

P1 : $(WRL_B(p_A) \rightarrow commit(t)) \;\vee\; (p_A \rightarrow commit(t))$

This protocol requires that the redo log is written or the operation p_A has been applied before the transaction commits (where transaction t issued operation p_A).[2] Here the event $commit(t)$ denotes the commit of transaction t.

Example: Undo Log

Writing the undo log on subsystem B for operation p guarantees it will be possible to erase the effects of p on A. This is done by applying p's inverse operation p_A^-, and is written:

G2 : $U(p_A) \in B \rightsquigarrow p_A^-$

Notice that we omit details about the meaning of p_A's inverse p_A^- in guarantee G2. Informally, we must require that the occurrence of p^- must restore the subsystem A to the state it would be in had p_A never occurred. This is spelled out in the ACTA formalism [3], but we do not consider it further in this paper.

We now specify the protocol that ensures that guarantee G2 will be applicable should a transaction need to be undone.

Protocol P2, also known as Write-ahead logging (WAL), requires that an undo log record for an operation be stored before the operation is applied:

[2] In the presence of delegation (see [3]), t is the transaction *responsible* for operation p.

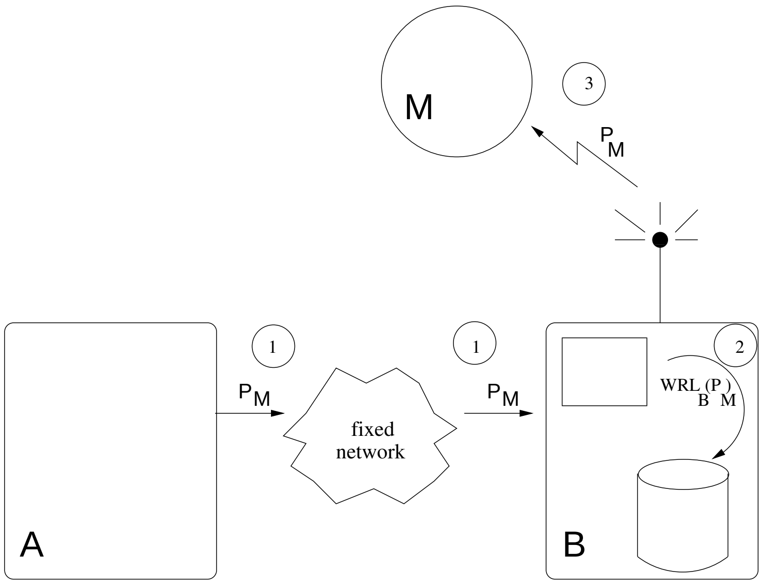

Figure 2: Host A sends operation p_M to mobile host M via B; B logs it before sending it on to M. (See History H1.)

P2 : $WUL_B(p_A) \rightarrow p_A$

WAL (P2) preserves the ability to undo the operation; notice that A and B may be the same subsystem.

These guarantees are supported by the recovery system, as follows: during normal (prevention) processing, the recovery system ensures that logs are made persistent and thus safe from failures; after a failure, during repair processing, the recovery systems accesses the logs and uses them to apply the undo as necessary.

3 Mobile System

We consider a mobile system described by Pradhan et al. [9], which consists of a set of fixed base hosts and mobile hosts. Each *base* covers a cell, which is an area in which there may be zero or more mobile hosts. A mobile host may move from one cell to another, but at any given time it communicates with at most one base host. For our purposes it is sufficient to consider a single mobile host, which we denote with M in the sequel, because we assume that mobile hosts do not interact directly for the purposes of recovery.

Distributed applications require exchange of messages between (local and mobile) hosts and user inputs at the mobile hosts. Those are the operations that change the state of a host. A message sent to a mobile host M is first sent to the base host B which covers the cell that M is in; B then forwards the message to M.

For recovery, the system uses stable storage on fixed hosts, because disk storage at a mobile host is frequently disconnected and deemed vulnerable to catastrophic failures.

In this paper we adopt the logging approach proposed in [9]: before a message is transmitted to a mobile host M, its base station host B logs it. Even inputs at M are sent to B and effected at M only after B's acknowledgment.

Before we introduce the protocols and guarantees specific to the mobile system, we illustrate the behavior of the system via a sample history, H1 (see also Figure 2):

H1 : $send_A(p_M, B),\ recv_B(p_M),\ WRL_B(p_M),$
$send_B(p_M, M),\ recv_M(p_M),\ p_M$

This says that (a transaction on) system A issues an operation p_M, which is sent to M's base host (denoted B). Then B logs p_M before sending it on to M, where it is received and applied.

Mobile System Notation

The following augments the notation of Section 2 for the mobile system case. Here the subsystems of interest are the hosts (fixed and mobile); we also introduce notation to describe the notions of base, handoff, and the connectivity between hosts.

- $A, B, C, ...$ denote hosts (subsystems); M denotes a mobile host. Predicate $Mov(A)$ is true if and only if host A is mobile.
- $Cnx(A, B)$ denotes that A and B are connected, i.e., that they can exchange messages with each other.
- $B = base(M)$ means M is within the cell controlled by host B (the base). Base hosts are always fixed.
- $send_A(p_C, B)$ denotes that host A sends a message to host B containing operation p, to be applied on host C. We write $send_A(p_A, A)$ when a local transaction invokes operation p on A.
- $recv_A(p_B)$ denotes that host A receives a message containing operation p, which must be applied on host B.

- $inpt(p_A)$ denotes that a user at A locally invokes operation p to be executed on A.
- $hndf_A(M, B)$ denotes that host A hands off mobile host M to host B, or in other words that M leaves A's cell and enters B's cell.[3]

Mobile System Properties

The following properties characterize the mobile system under study. Direct communication is always possible for any pair of *fixed* hosts. A mobile host can only communicate with its base host. Specifically:

- A fixed host is connected with all other fixed hosts: $\neg Mov(A) \Rightarrow (\forall B, \neg Mov(B) \Rightarrow Cnx(A, B))$
- A mobile host is directly connected to its base host (if it is connected at all): $Mov(M) \Rightarrow ((Cnx(M, B) \Rightarrow B = base(M))$ and B is unique)
  $(base(M) = B) \Rightarrow Cnx(M, B)$
- Each *receive* always has its corresponding *send*: $send_A(p, B) \rightarrow recv_B(p)$

We omit notation for the transitive closure of Cnx but it is obvious that when a mobile host M is in a cell it can communicate with any fixed host via its base.

Protocols and Guarantees

Let us now consider the protocols and guarantees of the mobile system described above. For brevity, we write only the redo version of the protocols and guarantees; the undo ones are symmetric. The first protocol specifies the origin of operations (P3). The other two (P4 and P5) prescribe the log-before-apply strategy.

An operation always originates from either a local input or an invocation (local when $A = B$, or from another host) by a transaction:

P3 : $(inpt(p_A) \rightarrow p_A) \vee (send_B(p_A, A) \rightarrow p_A)$

When a base host receives an operation addressed to a mobile host within its cell, it logs it before forwarding it:

P4 : $WRL_{base(M)}(p_M) \rightarrow send_{base(M)}(p_M, M)$

Before an operation may be applied at the mobile host M, it must be logged at M's base:

P5 : $WRL_{base(M)}(p_M) \rightarrow p_M$

We now consider the guarantees aspect of the system. Before an operation is applied at M it must be logged at $base(M)$, which guarantees that the operation will be reproducible later if necessary. The guarantee is:

G3 : $\{R_(p_M) \in base(M) \wedge Cnx(base(M), M)\} \rightsquigarrow p_M$

Recall that one of the properties of the mobile system is $(base(M) = B) \Rightarrow Cnx(M, B)$, thus the second clause of the guarantee's antecedent always holds.

Guarantee G3 with protocols P4 and P5 capture the notion that the base station is expected to keep recovery information for the mobile host. Guarantee G3 is an instance of the more general redo guarantee G4, which states that if the information to effect operation p on host B is available at host A, and A and B can communicate, then p_B is guaranteed:

G4 : $\{R(p_B) \in A \wedge Cnx(A, B)\} \rightsquigarrow p_B$

3.1 Handoffs

The preceding does not take into account the effect on recovery of a mobile host's migration through the network. Migration is represented by a *handoff*, which happens when a mobile host M leaves a cell with base A and enters a new cell whose base is B. We require that recovery-related information for M initially available on A must still be available should recovery be necessary while M is in B's cell.

We first discuss the handoff at an abstract level. M's migration from A to B causes A's guarantees to M to be honored by B because of (i) the guarantees A gave M, and (ii) a guarantee between all fixed hosts on the network (given by their connectivity), in particular between A and B. Thus the guarantees necessary to M are inferred from the guarantees between M and A and the connectivity guarantee between any two fixed hosts (A and B). For brevity we write only the redo cases; the undo are similar.

Guarantees between fixed hosts

The system guarantees between fixed hosts in the network say that if one of them has redo (or undo) recovery information for an operation, the other can obtain it too:

G5 : $\{\neg Mov(A) \wedge \neg Mov(B) \wedge R(p) \in A\} \rightsquigarrow recv_B(R(p))$

These guarantees are supported by the architecture of the fixed host network; the connectivity, and some network management software, ensure that station B can obtain the recovery information from station A. The specification of *when* (or if) that information is transferred is done via protocols that prescribe what must be done at a handoff, and for which we specialize the guarantee G5 by replacing the antecedent with an implementation of how to record and convey recovery information. In the sequel we deal with redo guarantees; undo guarantees are similar.

Eager and Lazy Handoffs

To make the recovery information available to a mobile host, the approaches are *eager* (pessimistic in [9]), in which the information follows M on the fixed network (i.e., as M moves from A to B, its recovery information is forwarded from A to B); and *lazy*, in which

[3] It is possible for M to leave A and be outside any other cell; we ignore that case.

information is only fetched from the original host if and when necessary (i.e., the information remains at A, and B just keeps track of that fact).

For the eager approach, protocol P6 simply ensures that when a mobile host arrives in a new cell, the previous base station has transmitted the recovery information to the current cell as part of the handoff:

P6 : $R(p_M) \in A \Rightarrow \{send_A(R(p_M), B) \rightarrow hndf_A(M, B)\}$

Because protocol P6 ensures that the information is sent at the time of handoff, the antecedent of G5 ($R(p) \in A$) holds. When recovery information resides on A (because p was invoked while M was in A's cell), as transmitting the recovery information to B is always possible (see Mobile System Properties), this means that the recovery information is always available at the current base host for M.

Implementing the lazy approach only requires that there exist a linked list of the base stations visited by the mobile host in which there is extant recovery information relevant to the host. One way to ensure that is logging the handoff events, so that recovery can later find the appropriate information.

P7 : $WRL_B(hndf_A(M, B)) \rightarrow hndf_A(M, B)$

For completeness, notice that when the recovery information was generated in the same cell where the mobile host is, satisfying the guarantee G5 is trivial, because we have the consequent already.

The preceding discussion of eager and lazy handoffs is interesting on several accounts. First, it shows how the same guarantee can be combined with two different protocols depending on the desired policy. Guarantee G5 describes a property of the architecture of the system, namely that the connectivity between fixed hosts facilitates propagating recovery information between them. Protocol P6 describes how to use the guarantee –propagate recovery information– eagerly; P7, how to do it lazily. Thus guarantees and protocols decompose recovery. An intuitive view of these characterizations is that protocols prescribe correct normal processing behavior for transactions (e.g., log before installing), whereas guarantees prescribe correct behavior for the underlying infrastructure, which must hold even through failures (e.g., fixed hosts are always connected).

Second, handoffs present a simple example of composition of guarantees. The recovery guarantee originally obtained by the mobile host in the first cell which logged its operations is revalidated as the mobile host moves. The new guarantee is derived as a composition of the original guarantee (between M and A in the terms of the preceding discussion) and the guarantee extant between fixed nodes in the network.

Third, the contrast between eager and lazy handoffs makes it clear that guarantees and protocols strive to maintain the recovery requirements necessary to make recovery *possible*. What guarantee G5 and protocols P6 or P7 ensure is that, after a crash, the recovery system will find sufficient information to enable it to repair the state of M. They do *not* ensure that the state of M will be correct after a failure – that is the responsibility of the repair phase, which we consider in subsection 3.2. Thus it becomes apparent that the repair phase of recovery, while it relies on the invariants induced by the protocols and guarantees of the normal processing phase, admits of separate treatment in terms of its own protocols and guarantees. We focus on crash and repair next.

3.2 Crash and Repair

The *repair phase* of recovery uses log information (possibly resident on various places) to reconstruct the state of the mobile host. The high-level property at work here is that the information necessary for undo or redo (as necessary) is durably stored in one of the hosts. In this subsection we briefly outline the protocols of the repair phase. One set of protocols delimits the repair actions, by prescribing which events must happen between the occurrence of a crash event and the completion of repair work. The other set of protocols describes the repair work itself, for example indicating that the effects of a 'loser' operation (e.g. one belonging to a transaction uncommitted before the crash) must be undone.[4]

The repair phase begins after a crash event. Thus we need to establish a protocol requirement that if a crash happened recovery must happen too. A committed operation (on M) must be *installed*[5] in M *and somewhere in the fixed network*, because we do not trust M's storage to be durable. Thus repairing the database will mean making the operation available on M (so M's state reflects a committed state) as well as somewhere else durable –a host on the fixed network, but not necessarily $base(M)$, as that's a policy choice. Thus we specify that for each operation p issued before the crash, repair must verify that p is committed but uninstalled, find the redo information for p; move the redo info for p to M; and using the guarantee that redo-info can be used to redo p, install p.

The repair phase ends when the state of the whole database has been restored to the committed projection of the history up to the crash. This may not be a single event; instead it is characterized on a per-object and per-operation basis.

Certain type of operations –repair work– can only happen while the database is being repaired; i.e., the events cannot appear at other times in the history of the system.

Also, the *order* in which operations are applied during repair work is important, both to recreate the state correctly and to improve the performance of recovery. Lomet and Tuttle [6, 7] identify the minimal order that the correct redo installation of operations during the repair work must follow to preserve correctness. The method we outline here for the formalization of the repair phase uses of the log information to ascertain the original order of the op-

[4] Besides indicating what are correct repair histories, the protocols yield important properties that are otherwise stated ad-hoc: for example, the property that the system will not unilaterally undo an operation is a consequence of the fact that this particular action in permitted only during repair.

[5] Installing an operation means making its effects permanent on the (stable) database.

erations, enabling the repair algorithms to abide by the restrictions identified by their work.

4 Related Work

Few researchers have dealt with the formalization of aspects of recovery. For example, [5] uses I/O automata to formally describe a recovery system based on ARIES; however, his description is at a low level of abstraction, close to the implementation.

Focusing on the redo portion of recovery, Lomet and Tuttle [6, 7] derive and prove the correctness of a redo recovery algorithm based on an installation graph that imposes an ordering significantly weaker than the usual concurrency control conflict graph. From this characterization they develop algorithms to manage the volatile storage, a test to choose which operations from the log must be redone, and an idempotent recovery algorithm that uses this test. In particular, they identify the weakest order required in the installation of operations during the repair phase of recovery. This is very useful in specifying correctness of recovery at the low-level of recovery, as well as leading to improved cache management. Our work complements theirs, because we describe higher-level properties (via guarantees and protocols) of the recovery subsystem whereas Lomet and Tuttle's prescribe how to preserve correctness at the level of installing updates.

5 Conclusions

In some sense, protocols and guarantees are very closely tied to each other. Whereas protocols talk about what must be done to achieve correct behavior, a guarantee states what can be expected if certain events occur or operations are performed correctly.

In this paper, we have shown that the abstractions of *recovery guarantees* can be very helpful in exposing the connections and expectations that exist between different components of a mobile system involved in a transactional activity. Similarly, *recovery protocols* make behavior requirements explicit.

By describing the mobile system and its behavior in terms of guarantees and protocols, we obtained the following benefits. First, we used abstraction to separate what a component can expect from another; e.g., the mobile host's (recovery) expectations of the fixed hosts. Also, the flip side of this abstract view serves to show what the system and each component must provide to others. Moreover, this documents precisely the challenges of providing recovery under mobility. For example, in showing handoff handling in terms of protocols and guarantees, we exposed assumptions that were implicit in the discussion of recovery [9]. We showed how the guarantees of mutual communication across the fixed network, composed with the original recovery guarantee for an operation, enable the recovery of operations once the mobile host has moved to a different cell.

Second, in our describing the alternatives for handoffs we used abstraction to characterize the broad requirement (a more abstract guarantee) all approaches must satisfy, and then precisely showed how that requirement is satisfied by the eager and the lazy approach. In further work we will use this approach of refining guarantees to precisely specify the recovery machinery, which in this paper we described informally.

We believe that because the abstractions of guarantees and protocols help in understanding what a component is expected to do, they will enable using a divide and conquer approach to crafting recovery protocols and subsystems. To this end, we are planning to examine the crafting of recovery for other transaction processing platforms and for workflow systems.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass., 1987.

[2] Luis-Felipe Cabrera, John A. McPherson, Peter M. Schwarz, and James C. Wyllie. Implementing Atomicity in Two Systems: Techniques, Tradeoffs and Experience. *IEEE Trans. on Software Engineering*, 19(10):950–961, October 1993.

[3] Panos K. Chrysantis and Krithi Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Trans. on Database Systems*, 10(3):450–491, September 1994.

[4] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, Calif., 1993.

[5] Dean Kuo. Model and Verification of a Data Manager Based on ARIES. *ACM Trans. on Database Systems*, 21(4):427–479, December 1997.

[6] David Lomet and Mark R. Tuttle. Redo recovery after system crashes. In *Proc. of the 21st Int'l Conf. on Very Large Data Bases*, pages 457–468, Zürich, September 1995.

[7] David Lomet and Mark R. Tuttle. Logical logging to extend recovery to new domains. In *Proc. of the 1999 ACM SIGMOD*, pages 73–84, Philadelphia, Penn., June 1999.

[8] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks using Write-Ahead Logging. *ACM Trans. on Database Systems*, 17(1):94–162, March 1992.

[9] Dhiraj K. Pradhan, P. Krishna, and Nitin H. Vaidya. Recoverable mobile environments: Design and trade-off analysis. Technical Report 95-053, Department of Computer Science, Texas A & M University, College Station, Texas, 1995.