M90] S. B. Zodnik and D. Maier. *Readings in Object-Oriented Database Systems.* Morgan Kaufmar
San Mateo, California, 1990.

GMA89] K. Salem, H. Garcia-Molina, and R. Alonso. Altruistic locking: A strategy for coping with long liv transactions. In D. Gawlick, M. Haynie, and A. Reuter, editors, *Lecture Notes in Computer Scienc High performance Transaction Systems*, volume 359, pages 175–199. Springer-Verlag, 1989. Al available as CS-TR-087-87, Computer Science Department, Princeton University. A more rece version appears as UMIACS-TR-90-104, University of Maryland Institute for Advanced Comput Studies.

ke82] D. Skeen. Non-blocking commit protocols. In *Proceedings of ACM-SIGMOD 1982 Internatio Conference on Management of Data, Orlando*, pages 133–147, 1982.

KPO88] M. R. Stonebraker, R. H. Katz, D. A. Patterson, and J. K. Ousterhout. The design of XPR In *Proceedings of the Fourteenth International Conference on Very Large Databases, Los Angel* pages 318–330, 1988.

KS91] N. R. Soparkar, H. F. Korth, and A. Silberschatz. Techniques for failure-resilient transaction ma agement in multidatabases. Technical Report TR-91-10, The University of Texas at Austin, Co puter Sciences Department, 1991. Submitted for publication.

LKS91] N. R. Soparkar, E. Levy, H. F. Korth, and A. Silberschatz. Adaptive commitment for real-ti distributed transactions. Forthcoming, May 1991.

on88] S.H. Son, editor. *ACM SIGMOD Record: Special Issue on Real-Time Databases.* ACM Press, Mar 1988.

S90] N.R. Soparkar and A. Silberschatz. Data-value partitioning and virtual messages. In *Proceedi of the nineth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Syste Nashville*, pages 357–367, April 1990.

n-91] Special issue on advanced transaction models. *Data Engineering*, 14(1), March 1991.

ei89] J. Veijalainen. *Transaction Concepts in Autonomous Database Environments.* R. Oldenbourg Verla Munich, 1989.

W90] J. Veijalainen and A. Wolski. The 2PC agent method and its correctness. Technical Report Resea Notes 1192, Technical research Centre of Finland, December 1990.

/ei88] W. E. Weihl. Commutativity-based concurrency control for abstract data types. *IEEE Transacti on Computers*, C-37(12):1488–1505, December 1988.

/ei89] W. E. Weihl. The impact of recovery on concurrency control. In *Proceedings of the Eighth AC SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia*, pag 259–269, 1989.

/HBM90] G. Weikum, C. Hasse, P. Broessler, and P. Muth. Multi-level recovery. In *Proceedings of the nin ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville*, pag 109–123, 1990.

/V90] A. Wolski and J. Veijalainen. 2PC agent method: Acieving serializability in presence of failures i heterogeneous multidatabase. In *Proceedings of the International conference on databases, para architectures and their applications (PARBASE)*, pages 321–330, March 1990.

IKN90] P. Muth, W. Klas, and E. Neuhold. How to handle global transactions in heterogeneous databa systems. Presented at the Workshop on Multidatabases and Semantic Interoperability, Tulsa, C lahoma, November 1990.

L89] C. Mohan and F. Levine. ARIES/IM: An efficient and high concurrency index management meth using write-ahead logging. Technical Report RJ6846, IBM Research, August 1989.

LO86] C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the R* distributed databa management system. *ACM Transactions on Database Systems*, 11(4):378–396, December 1986.

oh89] C. Mohan. ARIES/KVL: A key-value locking method for concurrency control of multiaction tra actions operating on B-trees indexes. Technical Report RJ7008, IBM Research, September 1989

oh90] C. Mohan. Commit-LSN: A novel and simple method for reducing locking and latching in tra action processing systems. In *Proceedings of the Sixteenth International Conference on Very La Databases, Brisbane*, 1990. A vesion available as IBM research report RJ 7344.

os87] J. E. B. Moss. Log-based recovery for nested transactions. In *Proceedings of the Thirteenth Int national Conference on Very Large Databases, Brighton*, pages 427–432, 1987.

P91] C. Mohan and H. Pirahesh. ARIES-RRH: restricted repeating of history in the ARIES transacti recovery method. In *Proceedings of the Seventh International Conference on Data Engineeri Kobe, Japan*, April 1991.

R91] P. Muth and T. C. Rakow. Atomic commitment for integrated database systems. In *Proceedings the Seventh International Conference on Data Engineering, Kobe, Japan*, April 1991.

ap86] C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Pre Rockville, Maryland, 1986.

KH88] C. Pu, G. Kaiser, and N. Hutchinson. Split-transactions for open-ended atctivities. In *Proceedi of the Fourteenth International Conference on Very Large Databases, Los Angeles*, pages 26– 1988.

L91a] C. Pu and A. Leff. Execution autonomy in distributed transaction processing. Technical Repo CUCS-024-91, Department of Computer Science, Columbia University, 1991.

L91b] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *Proceedi of ACM-SIGMOD 1991 International Conference on Management of Data, Denver, Colorado*, pag 377–386, May 1991.

ELL90] M. E. Rusinkiewicz, A. K. Elmagarmid, Y. Leu, and W. Litwin. Extending the transaction mo to capture more meaning. *SIGMOD Record*, 19(1):3–7, March 1990.

eu82] A. Reuter. Concurrency on high-traffic data elements. In *Proceedings of the ACM SIGAC SIGMOD Symposium on Principles of Database Systems, Los Angeles*, pages 83–92, 1982.

eu89] A. Reuter. ConTracts: A means for extending control beyond transcation boundaries. Presentati at 3rd Workshop on High Performance Transaction Systems, Pacific Grove, CA, September 198

M89] K. Rothermel and C. Mohan. ARIES/NT: A recovery method based on write-ahead loggi for nested transactions. In *Proceedings of the Fifteenth International Conference on Very La Databases, Amsterdam*, 1989. A longer version is available as IBM Research report RJ 6650 (6396

S88] H. F. Korth and G. Speegle. Formal model of correctness without serializability. In *Proceedings ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 379–38 June 1988.

S90] H. F. Korth and A. Silberschatz. *Database System Concepts.* McGraw-Hill, New York, 1990. Seco Edition.

am78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications the ACM*, 21(7):558–565, July 1978.

am81] B. W. Lampson. Atomic transactions. In *Lecture Notes in Computer Science, Distributed Syste — Architecture and Implementation: An Advanced Course*, pages 246–265. Springer-Verlag, Berl 1981.

C87] T. J. Lehman and M. J. Carey. A recovery algorithm for a high-performance memory-reside database system. In *Proceedings of ACM-SIGMOD 1987 International Conference on Manageme of Data, San Francisco*, pages 104–117, 1987.

n80] B. G. Lindsay. Single and multi-site recovery facilities. In *Distributed Databases*, chapter 10, pag 247–284. Cambridge University Press, Cambridge, U.K., 1980. Also available as IBM Resear Report RJ2571, San Juse, July, 1979.

KS91a] E. Levy, H. F. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transacti management. In *Proceedings of ACM-SIGMOD 1991 International Conference on Management Data, Denver, Colorado*, pages 88–97, May 1991.

KS91b] E. Levy, H. F. Korth, and A. Silberschatz. A theory of relaxed atomicity. In *Proceedings of the AC SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1991. To appear

YI87] Y.-H. Lee, P. S. Yu, and B. R. Iyer. Progressive transaction recovery in distributed DB/DC systen *IEEE Transactions on Computers*, C-36(8):976–987, August 1987.

yn83] N. Lynch. Multi-level atomicity. *ACM Transactions on Database Systems*, 8(4):484–502, Decemb 1983.

ap89] Multidatabase services on ISO/OSI networks for transactional accounting. Technical Repo MAP761B, SWIFT, INRIA, GMD/FOKUS, University of Dortmund, 1989. Final Report, Edit by S.W.I.F.T. Society for Worldwide Interbank Financial Telecommunications s.c. 81 avenue Erne Solvay, B-1310 La Hulpe, Belgium.

D89] D. R. McCarthy and U. Dayal. The architecture of an active data base management system. *Proceedings of ACM-SIGMOD 1989 International Conference on Management of Data, Portlar Oregon*, pages 215–224, June 1989.

GG86] J. E. B. Moss, N. D. Griffeth, and M. H. Graham. Abstractions in recovery management. *Proceedings of ACM-SIGMOD 1986 International Conference on Management of Data, Washingto* pages 72–83, 1986.

HL$^+$90] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A transaction recove method supporting fine-granularity locking and partial rollbacks using write-ahead logging. Tec nical Report RJ 6649 (63960), IBM Research, February 1990. A revised vesion. To appear in AC Transactions on Database Systems.

db90] Special issue on heterogeneous databases. *ACM Computing Surveys*, 22(3), September 1990.

er90] M. Herlihy. Apologizing versus asking permission: optimistic concurrency control for abstract da types. *ACM Transactions on Database Systems*, 15(1):96–124, March 1990.

MS88] R. Haskin, Y. Malachi, and W. Sawdon. Recovery management in QuickSilver. *ACM Transactio on Computer Systems*, 6(1):82–108, February 1988.

R83] T. Haerder and A. Reuter. Principles of transaction oriented database recovery — a taxonon *ACM Computing Surveys*, 15(4):289–317, December 1983.

R87] T. Haerder and K. Rothermel. Concepts for transaction recovery in nested transactions. In *P ceedings of ACM-SIGMOD 1987 International Conference on Management of Data, San Francis* pages 239–248, 1987.

S91] M. Hsu and A. Silberschatz. Unilateral commit: a new paradigm for reliable distributed transacti management. In *Proceedings of the Seventh International Conference on Data Engineering, Ko Japan*, pages 286–293, April 1991.

oh90] W. Johannsen. Transaction models for federative distributed database systems. In *Proceedings the International Conference on Information Technology (InfoJapan 90)*, pages 285–292, 1990.

B91] N. Krishnakumar and A. J. Bernstein. Bounded ignorance in replicated systems. In *Proceedi of the tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systen Denver*, pages 63–74, May 1991.

KB88] H. F. Korth, W. Kim, and F. Bancilhon. On long duration CAD transactions. *Information Scienc* 46:73–107, October 1988.

LMP84] W. Kim, R. Lorie, D. McNabb, and W. Plouffe. Nested transactions for engineering design databas In *Proceedings of the Tenth International Conference on Very Large Databases, Singapore*, pag 355–362, 1984.

LS90a] H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transa tions. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisba* pages 95–106, August 1990.

LS90b] H. F. Korth, E. Levy, and A. Silberschatz. An optimistic two-phase commit protocol. Technical F port TR-90-31, The University of Texas at Austin, Computer Sciences Department, 1990. This wo was presented in the Workshop on Multidatabases and Semantic Interoperability, Tulsa, Novemb 1990.

oo90] G.M. Koob, editor. *Foundation of Real-Time Computing Research Initiative.* Office of Naval search, October 1990. Third Annual Workshop.

or83] H. F. Korth. Locking primitives in a database system. *Journal of the ACM*, 30(1):55–79, Janua 1983.

R88] J. Klein and A. Reuter. Migrating transactions. In *Future Trends in Distributed Computer Syste in the '90s, Hong Kong*, 1988.

uc90] D. Duchamp. Analysis of transaction management performance. In *Proceedings of the Twelfth AC Symposium on Operating Systems Principles, Litchfield Park, Arizona*, pages 177–190, Decemb 1990.

D89] A. K. Elmagarmid and W. Du. Supporting value dependencies for nested transactions in interba Technical Report CSD-TR-885, Purdue University, Computer Sciences Department, May 1989.

LLR90] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. E. Rusinkiewicz. A multidatabase transaction mo for InterBase. In *Proceedings of the Sixteenth International Conference on Very Large Databas Brisbane*, pages 507–518, 1990.

V87] F. Eliassen and J. Veijalainen. An S-transaction definition language and execution mechanis Technical Report 275, GMD—Gesellschaft fur Mathematic und Datenverarbeitung MBH, Novemb 1987.

C87] R. S. Finlayson and D. R. Cheriton. Log files: An extended file service exploiting write-once stora In *Proceedings of the 11th Symposium on Operating Systems Principles*, pages 139–148, 1987.

b87] Special issue on federated databases systems. *Data Engineering*, 10(3), September 1987.

O89] A. A. Farrag and M. T. Ozsu. Using semantic knowledge of transactions to increase concurren *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

LPT75] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. Granularity of locks and degrees of cons tency in a shared data base. In *IFIP Working Conference on Modeling of Data Base Manageme Systems*, pages 1–29, 1975. Also available as Research Report RJ1654, IBM, September 1975.

M$^+$81] J. N. Gray, P. McJones, et al. The recovery manager of the system R database manager. *AC Computing Surveys*, 13(2):223–242, 1981.

M83] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed databa *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

MAB$^+$83] H. Garcia-Molina, T. Allen, B. Blaustein, R. M. Chilenskas, and D. R. Ries. Data-patch: Integrati inconsistent copies of a database after a partition. In *Third IEEE Symposium on Reliability Distributed Software and database Systems*, pages 38–48, 1983.

MGK$^+$90] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating multi-transacti activities. Technical Report UMIACS-TR-90-24, University of Maryland Institute for Advanc Computer Studies, February 1990.

MS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of ACM-SIGMOD 1987 Internatio Conference on Management of Data, San Francisco*, pages 249–259, 1987.

ra78] J. N. Gray. Notes on database operating systems. In *Lecture Notes in Computer Science, Operati Systems: An Advanced Course*, volume 60, pages 393–481. Springer-Verlag, Berlin, 1978.

ra80] J. N. Gray. A transaction model. In *Lecture Notes in Computer Science, Automata Languages a Programming*, pages 282–298. Springer-Verlag, Berlin, 1980.

ra81] J. N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Seve International Conference on Very Large Databases, Cannes*, pages 144–154, 1981.

R90] B. R. Badrinath and K. Ramamritham. Performance evaluation of semantics-based multilevel co currency control protocols. In *Proceedings of ACM-SIGMOD 1990 International Conference Management of Data, Atlantic City, New Jersey*, pages 163–172, 1990.

S88] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceedings of ACM-SIGM 1988 International Conference on Management of Data, Chicago*, pages 135–141, 1988.

ST90] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Reliable transaction management in a mul database system. In *Proceedings of ACM-SIGMOD 1990 International Conference on Manageme of Data, Atlantic City, New Jersey*, pages 215–224, 1990.

SW88] C. Beeri, H.-J. Schek, and G. Weikum. Multi-level transaction management, theoretical art practical need? In *International Conference on Extending Database Technology, Lecture Notes Computer Science*, volume 303. Springer Verlag, 1988.

$^+$89] S. Chakravarthy et al. HiPAC: A research project in active, time-constrained database manageme Technical Report XAIT-89-02, Xerox Advanced Information Technology, July 1989. Final repor

KKS89] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The case for safe RAM. In *Proceedings the Fifteenth International Conference on Very Large Databases, Amsterdam*, pages 327–336, 19

P87] S. Ceri and G. Pelagatti. *Distributed Database Systems, Principles and Systems.* McGraw-Hill, N York, 1987.

R90] P. K. Chrysanthis and K. Ramamritham. ACTA: A framework for specifying and reasoning ab transaction structure and behavior. In *Proceedings of ACM-SIGMOD 1990 International Confere on Management of Data, Atlantic City, New Jersey*, pages 194–203, 1990.

R91] P. K. Chrysanthis and K. Ramamritham. A formalism for extended transaction models. In *P ceedings of the Seventeenth International Conference on Very Large Databases, Barcelona*, 1991.

av73] C. T. Davies. Recovery semantics for a DB/DC system. In *Proceedings of the ACM Ann Conference, Atlanta*, pages 136– 141, 1973.

E89] W. Du and A. K. Elmagarmid. Quasi serializability: a correctness criterion for global conc rency control in InterBase. In *Proceedings of the Fifteenth International Conference on Very La Databases, Amsterdam*, pages 347–355, 1989.

HL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactio In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlan City, New Jersey*, pages 204–214, 1990.

KO$^+$84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, and M. R. Stonebraker. Implementati techniques for main memory database systems. In *Proceedings of ACM-SIGMOD 1984 Internatio Conference on Management of Data, Boston*, pages 1–8, 1984.

ST87] D. S. Daniels, A. Z. Spector, and D. S. Thompson. Distributed logging for transaction processi In *Proceedings of ACM-SIGMOD 1987 International Conference on Management of Data, S Francisco*, pages 82–96, 1987.

Bibliography

A90] D. Agrawal and A. El Abaddi. Locks with constrained sharing. In *Proceedings of the nineth AC SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville*, pages 8 93, April 1990.

AL91] D. Agrawal, A. El Abaddi, and A. E. Lang. Performance characteristics of protocols with order shared locks. In *Proceedings of the Seventh International Conference on Data Engineering, Ko Japan*, April 1991.

GMS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedur in federated database systems. *Data Engineering*, 10(3):5–11, September 1987.

BG89] C. Beeri, P. A. Bernstein, and N. Goodman. A model for concurrency in nested transaction systen *Journal of the ACM*, 36(2):230–269, April 1989.

HB90] P. A. Bernstein, M. Hsu, and Mann B. Implementing recoverable requests using queues. In *Pr ceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic Ci New Jersey*, pages 112–122, 1990.

HG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databa Systems.* Addison-Wesley, Reading, MA, 1987.

it86] D. Bitton. The effect of large main memory on database systems. In *Proceedings of ACM-SIGMO 1986 International Conference on Management of Data, Washington*, pages 337–339, 1986. A par session, whose participants were Bitton, D. (chairperson), Garcia-Molina, H., Gawlick, D., a Lomet, D.

jo73] L. A. Bjork. Recovery scenario for a DB/DC system. In *Proceedings of the ACM Annual Conferen Atlanta*, pages 142–146, 1973.

OH+91] A. Buchmann, M. T. Ozsu, M. Hornick, D. Georgakopolous, and F. A. Manola. A transacti model for active distributed object systems. In A. K. Elmagarmid, editor, *Advanced Transacti Models for new applications.* Morgan-Kaufmann, 1991.

R87] B. R. Badrinath and K. Ramamritham. Semantic-based concurrency control: Beyond commu tivity. In *Proceedings of the Third International Conference on Data Engineering, Los Angel* 1987.

R88] B. R. Badrinath and K. Ramamritham. Synchronizing transactions on objects. *IEEE Transactio on Computers*, 35(5):541–547, May 1988.

Chapter 10

Conclusions

e summarize by listing the primary contributions of this dissertation:

- Giving substance and formal meaning to the notions of compensation and relaxed atomicity (i.e., the wo on $\mathcal{R}$-atomicity). Based on the formal results, a methodology for the design of compensating transactions envisioned. In light of the abundance of work that relies on semantic atomicity and compensation witho giving them specific meaning, we consider this contribution a significant one.

- The power and utility of semantics-based recovery were illustrated in the context of distributed transa tion management. With the aid of these methods we devised protocols that alleviate the inherent a hard problems that are associated with atomicity in distributed systems (i.e., the polarized and the O2I protocols).

- Using the compact model of composite transactions with polarities, we have identified a correctness criteri (namely isolation of recoveries) in the realm of transactions that are not atomic in the standard sense. Bas on the duality of compensation and retry, the criterion applies when both of these semantics-based metho are employed. This work helps understand how relaxing atomicity of a transactional unit interacts wi isolation of concurrent transactions.

The significance of the work on relaxed atomicity is underlined in light of the inevitable problems that a pical of atomicity in a distributed system. Moreover, relaxed atomicity is motivated by the growing interest stributed system integration; an area where standard atomicity stands in sharp contrast to the crucial autonor the integrated components.

The criterion of recovery isolation (IR) gives transactions a degree of isolation from inconsistencies arisi m failures and their asynchronous recoveries. In an IR execution, effects of both committed and abort btransactions of the same transaction are allowed to be exposed, thereby avoiding the prohibitive cost of stributed atomic commitment. However, it is ensured that transactions observe only effects of sets of steps wi entical polarity, thus hiding the non-atomic execution of transactions.

Finally, we point out that the ideas reported here constitute a bottom-up approach to an important pro m. The problem is the inability of the traditional transaction model to accommodate the demands of advanc tabase applications and environments. The solution we propose is semantics-based recovery. We have dev ed the solution in this dissertation in a step-wise manner. First, we have defined and studied the concept mpensation in a simple setting of a single transactional unit. Having done that, we have used compensation a ilding block in constructing more complex and structured transactions, and for solving problems in distribut vironments. Thus, we have demonstrated that semantics-based recovery mechanisms are useful in pointing c w solutions for the problems posed by advanced database systems.

- Formal development of the retry method is lacking. In conducting this research, it might be interesti to capitalize on the duality with compensation, where applicable. Again, the impact of the shape of t forward transaction on the semantics-based recovery, retry this time, will play a key role.

- An interesting trade-off between the complexity of the marking scheme and the degree of concurren allowed by the corresponding protocol is evident from the range of IR-preserving protocols that have be devised. Some parameters of this trade-off are summarized in Section 5.6. Interestingly, it seems that t inherent blocking phenomenon is manifested in the IR context by the difficulty to discard markers. A resu should qualify the complexity of obtaining IR and relate this complexity to the known results on atomic in distributed systems. Additionally, it would be interesting to relate our work on composite transactio and IR to the work on epsilon-serializability [PL91b, PL91a] and bounded ignorance [KB91]. Such a stu will shed light on the common denominator of trading transaction properties in a controlled manner f improved distributed transaction management.

- A more precise characterization of sensitive transactions is imperative. This definition should be a seman one, and as such it should complement the rather syntactic character of the IR criterion. The differenc between global and local consistency in a distributed database [DE89] are bound to surface when formalizi the notion of sensitivity. Once this definition is accomplished, one should look into minimizing overhead f enforcing IR for sensitive transactions when they are executed concurrently with non-sensitive transactio A clearer understanding of IR itself bears on sensitivity, too. It would be nice to gain deeper insig regarding the applicability of the two versions of IR presented in Chapters 6 and 7. The difference in t visibility propagation (i.e., the difference in the transitivity of the *follows* relation) are the crux for th matter.

- Applying the O2PC protocol in multidatabases requires additional work. We raise the following points:

  - It must be possible to distinguish between local and global transactions in order to let local transacti benefit from the released locks in case the simple scheme described in the beginning of Chapter 7 employed.

  - Some modifications to the lock manager software seem inevitable in order to support enforcing I However, since the interface to the lock manager and the two-phase locking rule are left intact, the modifications might be best accommodated by adding a software layer rather than actually modifyi the code of the lock manager as was outlined in Section 7.4.5.

  - As the protocol stands now, transactions that do not wish to access locally committed data that not globally committed, cannot do so as data items are unlocked once they are locally committed. I adding another operation with the appropriate semantics (like the 'release' operation of [SGMA89]) the lock manager interface and use this operation rather than unlocking the data item, these transa tions can be accommodated. The penalty is changing the interface of the locking manager.

  - Facilitating compensation by the local log (see Section 4.1.1) may not be allowed in multidatabas The alternative is to maintain a separate source of semantic information on the execution to gui compensation. However, again, because of autonomy concerns, actions of local transactions will n be recorded in such an external log. The more appealing scheme is that of using compensation federated databases, where a semantic decomposition of transaction is used, and compensation can relatively independent of the history.

Chapter 9

Future Research

he following list of topics are proposed as research that should augment the work reported in this dissertatio he topics are divided according to the structure of the dissertation.

.1 Single-Transaction Recovery

veral open problems were posed in Chapter 4. In what follows, we add a few more issues:

- Our compensation methodology should be refined by being tested with more example applications. Th experience should be used to reinforce the design methodology sketched in Section 4.2 and provide n insights in this respect. Sample specific problems are extracting relations $\mathcal{R}$ from consistency constrain and defining compensation based on such relations.
- In our treatment of $\mathcal{R}$-atomicity we left the relation $\mathcal{R}$ under-specified to a large extent. There is scope investigate the relationship between the shape of this relation and the corresponding atomicity notions. V have actually made the first strides in this direction by mentioning reflexive, anti-symmetric, and transiti relations, however more specific results are needed. It might be instructive to compare the notions recoverability, failure-commutativity, and in particular Herlihy's invalidation [Her90] with different forms $\mathcal{R}$-commutativity. Some initial results on relating these notions are reported in [CR91].
- As was mentioned in Section 4.2, exposing uncommitted data should be done in a qualified manner, bas on the properties of the exposing and exposed transactions. In principle, this observation calls for classifyi transactions into transactions types, and allowing exposing updates early only among compatible classes transactions. Work on compatibility of transaction classes can be found in [GM83].

.2 Atomicity of Composite Transactions

veral open problems were posed in Section 6.4. In what follows, we add a few more issues:

- The polarized protocol, as well as the protocols of Chapter 7 are given assuming a generic type of access data items. These protocols should be extended for the read/write case.
- More specific relationships between shapes of global transactions and the applicability of the localizati of compensation principle is requisite. A clearer classification of the issue of inter-dependencies amo subtransactions and its ramifications on relaxed atomicity and IR deserves more attention. In particul one should investigate when each of the two versions of IR we have presented is applicable.

ther the weaker criterion of *quasi serializability*. The authors of [AGMS87] mentioned use of compensati nsactions to cope with exposing updates to local transactions, and commit dependencies and cascading abor the recovery approach among global transactions.

In [DE89], a correctness criterion that is weaker then serializability is given for transaction manageme multidatabases. The treatment of dependencies among subtransactions in this paper, and the relaxation ializability is relevant to our work.

The methods reported in Chapter 6 and 7 are characterized by using semantic information to overcome t ficulties associated with the distributed commit problem. This characterization also suits ideas of [GMAB$^+$8 ere semantic information and compensatory actions are used to reconcile inconsistencies in a distribut tabase after a partition.

A different approach for solving the problem atomicity in distributed systems is based on the notion of sing e transactions [SS90, HS91]. The idea is to circumvent the problems of committing a multi-site transaction grating data to a single site and executing a local transaction. Mechanisms for reliable message transmissic relied upon for these types of schemes.

Independently of our work on establishing relaxed atomicity by semantics-based recovery, [MR91, MKN oort on how the traditional redo/undo methods are used to obtain standard atomicity in multidatabases.

ociety for Worldwide Interbank Financial Telecommunications) network, that is documented in [EV87, map8 i89]. The essential property of this international banking environment is that the component systems are a nomous. The system employs a semantic transaction (S-transactions) model. In this model, ACID transactio e used as building blocks in a similar manner to composite transactions by combining them with a control flo echanism.

Another enhancement to the transaction model is the split-transaction operation [PKH88]. Performing a sp eration, a transaction modeling an open-ended activity, commits data that will not change. Interactions wi her transactions are serialized through the committed data.

Regarding fitting non-compensatable actions into our ideas, we mention that [ELLR90] presents a transacti odel which distinguishes among compensatable and not compensatable subtransactions. A *mixed* transacti defined to be a global transaction where some of its subtransactions are compensatable and some are not.

In [CR90, CR91], a generic framework, called ACTA, is constructed for the specification and reasoning abo variety of transaction models. This framework can be instantiated to express existing models by defining a ri of attributes like visibility of effects, delegation of objects among transactional units, etc.

In the spirit of relaxing the classical transaction properties, we mention the work of [KB91]. This wo roduces a notion of bounded violation of consistency constraints in favor of increased concurrency. The bou based on the semantics of the constraints. The violation occurs as transactions are allowed to be ignorant ects of a bounded number of prior transactions. A similar idea is found in the work on epsilon-serializabil L91b]. There, temporal and bounded inconsistencies among replicas are allowed to be observed by transactio

There are very god reasons for relaxing the classic transaction model. However, in doing so, care should ken regarding the interactions among the relaxed properties and other properties. Some form of correctne terion must be defined and retained, given a new model. Most of the formerly mentioned work lacks in th spect. Only recently [PL91b, PL91a, KB91], some work has been devoted to correctness issues. The focus th these papers, however, is on relaxing concurrency control aspects.

Our work on composite transactions is not yet another advanced transaction model. We see our ma ntribution as the formulation of the correctness criterion (IR) and the corresponding protocols. IR captur th compensation and retry and deals with executions that are potential in most of the mentioned advanc nsaction models.

.4 Other Related work

ithin the class of serializable executions, some advances that are related to our work have been made recent [AA90], a locking-based protocol that captures the entire class of conflict serializability is reported. The perfo ance of this protocol is examined in [AAL91]. In [SGMA89], an extension to two-phase locking, called *altruis king*, is introduced as means for permitting release of locks held by long-duration transactions before they co t, while ensuring serializability. Transactions that access released but uncommitted data are said to be runni *the wake* of the releasing transaction and must abide by certain locking and committing restrictions in ord ensure database consistency. The major alternative proposed for recovery in [SGMA89] is the maintenan commit dependencies and executing cascading aborts in case that the releasing transaction is aborted. ernative for recovery based on compensation was also mentioned there, but not fully explored. The protoc [AA90, SGMA89] allow more concurrent executions compared to the 2PL protocol, while ensuring serializab . The protocols resemble our marking-based protocols in the manner they enforce certain orderings amo nsactions.

In [AGMS87], a variation of altruistic locking was proposed in the context of multidatabases. It was sho [DE89] that this particular variation of altruistic locking in multidatabases does not ensure serializability, b rrect concurrent execution lies with the concurrency control protocol. Standard **read** and **write** operatic e used in this model. By letting subtransactions execute in parallel, and by preserving the predicates explicit ncurrency is enhanced, yet correctness is guaranteed.

Another way for enhancing concurrency is the use of semantically-richer operations instead of the primiti **ad** and **write**. Having semantically-richer operations provides the means for refining the notion of conflicti rsus commutative operations [BR87, BR90, BR88, Wei88, Wei89]. That is, it is possible to examine wheth o operations commute (i.e., do not conflict) and hence can be executed concurrently. By contrast, in t nventional model, there is not much scope for such considerations since a write operation conflicts with a her operation on the same data item. For example, object-oriented databases use abstract data type techniqu define data objects which support specific and rather complex operations (see, e.g., [ZM90]). In [BR88], dition to using semantics of operations, the authors use the structure of complex objects to enhance concurren ing the concept of a granularity graph to represent the 'contained-in' relation, compatibility of operations termined dynamically, at run-time.

The transaction model we propose (see Section 2.2) can be viewed as a synthesis of the NT/PV model wi mplex operations and other means for embedding semantics within the model. The work in [Wei88, Wei ould be cited for its study of the subtle interplay among recovery and concurrency control issues.

An alternative paradigm of defining non-serializable, yet correct, executions is to refine the transaction boun es by prescribing breakpoints in transactions and by specifying allowable interleavings at these breakpoi yn83, FO89]. These specifications are based on semantic knowledge. Our IR criterion can be thought of as stance of this paradigm.

A major deficiency of most of the formerly mentioned work is that in the quest for alternative correctne teria, 'enhancing concurrency' was emphasized while disregarding transaction failures and recoveries.

.3 Advanced Transaction Models

ecently, a substantial amount of work has been dedicated to advanced transaction models (refer to [tm-91] fo oad spectrum of such models). The motivation for this research stems from a practical need to relax the clas omicity, Consistency, Isolation, and Durability (ACID) properties of transactions. Specific reasons for t end were already mentioned in the introduction (e.g., support for long-duration and cooperative transactio d autonomy concerns for multidatabases). It is common to exploit the semantics of data and activities for t nstruction of applications under these models. Our work on relaxed atomicity belongs to this trend.

Common in a few of the papers on advanced transaction models [GMGK$^+$90, KR88, Reu89] and in c n work, is the following abstraction of a complex transaction (we refer to such a transaction as a compos nsaction). A transaction is a collection of ACID subtransactions, each executing a logically coherent task, a llectively representing a complex and possibly, long-lived activity. A script (or work-flow) controls the invocati these subtransactions. In essence, this abstraction attaches a control flow structure to a set of transactio its. Typically, in the domain of such composite transactions it is assumed that serializability is ensured only e subtransaction level. It is implicitly assumed, that either a semantic criterion (not serializability) is enforc the level of entire transactions [KR88, Reu89], or no constraints at all are imposed at this level [GMGK$^+$9 so, the concepts of semantic atomicity and forward recovery are advocated in the context of these mode rward recovery is the capability to resume the execution of a failed transaction rather than aborting it. T operty can be obtained by using a subtransaction, rather than the entire transaction, as the unit of recovery. echanism for maintaining persistent linkage among subtransactions, (e.g., reliable queues [BHB90]) is essent the purposes of forward recovery.

A prominent example of an actual system that incorporates ideas that are similar to our work is the S.W.I.F.

nnot voluntarily abort itself is introduced. Low-level details of how to store reliably the code of compensati
nsactions, and record their identity in the log records of the saga's subtransactions are also discussed there

A major source of influence on our work was the study of multi-level transactions [BSW88, WHBM90, BBG8
particular we cite [BSW88], where several common ad hoc techniques in transaction management (e.g., ea
ease of locks on pages) are cast in terms of the elegant framework of multi-level transactions.

In [BR87], semantics of operations on abstract data types are used to define *recoverability*, which is a weak
tion than commutativity. Conflict relations are based on recoverability rather than commutativity. Con
ently, concurrency is enhanced since the potential for conflicts is reduced. When an operation is recovera
th respect to an uncommitted operation, the former operation can be executed; however a commit dependen
forced between the two operations. This dependency affects the order in which the operations should comm
they both commit. If either operation aborts, the other can still commit, thereby avoiding cascading abor
r instance, an invoked write operation is recoverable relative to an uncommitted read operation on the sa
ta item.

Recoverability-based conflict resolution for multi-level transactions is reported in [BR90]. There, simulatio
dicate that a recoverability-based multi-level scheme outperforms both single-level 2PL and commutativ
sed multi-level concurrency control.

A noteworthy approach, which can be classified as a simple type of compensation, is employed in the XP
stem [SKPO88]. There, a notion of *failure commutativity* is defined for entire transactions (as opposed
dividual operations). Failure commutativity is an adaptation of recoverability [BR87] applied to compl
nsactions. Transactions that are classified as failure commutative can run concurrently without any conflic
ndling the abort of such a transaction is done by a log-based special undo function, which is a special case
mpensation as we define it.

This type of work [BR87, BR90, SKPO88] is more conservative than ours as it relies on commit dependenc
d as it narrows the domain of interest to serializable histories. Our work starts with a different premise a
jective, as we explicitly allow and handle situations of exposed dirty data, and offer the extra flexibility
dressing such cases when the need arise. Our results offer several notions that are applicable in the wi
main that includes non-serializable and non-recoverable (as defined in [BHG87]) histories.

.2 Beyond Serializability

ring different stages of our work, we were influenced by studies of correctness criteria other than serializabili
is is evident primarily in Chapter 2, where a flexible model for dealing with non-serializable executions w
nstructed. The impact of the work on alternative correctness criteria is felt also in our treatment of distribut
nsaction management, where serializability is assumed only locally, and not as a global property. In this secti
review the sources of this impact on our work.

In order to deal with enhanced concurrency, beyond the realm of serializable executions, a new approach
ncurrency control is required. A major source of influence on our work in this respect is the NT/PV mo
scribed in [KS88]. Within this model, alternative correctness notions, other than serializability, can be define
e aspect of the NT/PV model that is of relevance to us are the use of explicit consistency predicates as means
pture the semantics of the database. Explicit *input* and *output predicates* over the database state are associat
th top-level transactions as well as with each nested transaction. The input predicate is a pre-condition
nsaction execution and must hold on the state that the transaction reads. The output condition is a po
ndition which the transaction guarantees on the database state at the end of the transaction provided th
ere is no concurrency and the database state seen by the transaction satisfies the input condition. Thus,
the standard model, when transactions are run in isolation, they preserve consistency, and responsibility

Chapter 8

Related Work

set of seminal papers on transaction and recovery management constitutes the background for this dissertati LPT75, Gra78, Lin80, Gra81, GM^{+}81, Lam81, HR83]. These articles shaped the attitude and understanding e author. The comprehensive article [MHL^{+}90] and several other of the ARIES papers [MP91, RM89, Moh ntributed to the understanding of the intricate issues of practical transaction management.

Some related work was mentioned in previous chapters, in a precise context. In this chapter, we menti earch that has impact on our own work, provides alternative approaches, or is related to issues raised in t ssertation. Work on compensation is reviewed in Section 8.1. Research on correctness notions other th ializability, and on advanced transaction models is covered in Section 8.2, and Section 8.3, respectively. her related work is included in Section 8.4.

.1 Compensation

e idea of compensating transactions as a semantically-rich recovery mechanism is mentioned, or at least referr in several papers. However, to the best of our knowledge, a formal and comprehensive treatment of the iss d its ramifications is lacking. Therefore, in light of the growing consensus for the need for compensat echanisms, we feel that our contribution in this respect is significant.

Strong motivation for our work can be found in Gray's early paper [Gra81]. There, compensating transactic e mentioned informally as 'post facto' transactions that are the only means to alter committed effects. Gr serves that early exposure of uncommitted data is essential in the realm of long-duration and/or nested transa ns. Also, compensation is mentioned as a possible remedy to the limitations of the current transaction mod other early reference is the DB/DC database system [Bjo73, Dav73], where the idea of semantic undoing ed.

The notion of compensation (countersteps) is mentioned in the context of histories that preserve consisten thout being serializable in [GM83, FO89]. It is noted in [GM83] that running countersteps (to undo ste es not necessarily return the database to its initial state, an observation on which we elaborate in our wo e difficulty of designing countersteps is raised as a drawback of compensation, which is another problem dress.

Compensating transactions are also mentioned in the context of a *saga*, a long-duration transaction that c broken into a collection of subtransactions that can be interleaved arbitrarily with other transactions [GMS8 saga must execute all its subtransactions, hence compensating transactions are used to amend partial executi sagas. In a saga, the last forward subtransactions to execute is simply rolled-back in case it aborts. Previc btransactions are compensated-for. In [GMS87] and in [GM83] the idea that a compensating transacti entire set. Multigranularity locking [GLPT75] would be very beneficial in this case since R1 and R2 requ locking of the entire set.

- In addition to protocols UD/LCUM and LC/UDUM there are a variety of other protocols resulting fro other isolation properties. For instance, a very simple protocol is one that requires that for each transacti T_s, all sites in which T_s executes are undone with respect to the same transactions, and are locally-committ with respect to no transaction. There is a trade-off between the protocol's simplicity and the degree concurrency it allows. Further details on the other protocols can be found in [KLS90b].
- Alternatively to storing the marking sets as data items in the database, they can be stored and manag externally. A special software module whose responsibility is the scheduling of global transactions wou maintain the marking sets. This module should implement a concurrency control scheme for accessing t marking sets. The concurrency control scheme can be customized to take full advantage of the sim access pattern to the marking sets. Such an architecture might be preferable in the multidatabase conte since storing the marking sets in the local database might be cumbersome and even prohibited. Typical in a multidatabase system, at each site, an *agent* [WV90, VW90] of the global transaction manager running as an application program, that is, above the local transaction manager. These agents spawn lo subtransactions, submit requests originating at the global transaction manager for local execution throu these subtransactions, and participate in the 2PC protocol as the representatives of their sites. The functio of managing the marking sets can be integrated into these agents.

Implementing UDUM1 may be cheaper in terms of messages. However, it requires augmenting the da uctures. Keeping track of the set of execution sites for each transaction is necessary. Also, it must be possil determine at what site a marking (T_j, UD) was observed by T_s. For brevity, we do not present here t cessary augmented data structures. We note, however, that managing these structures does not incur a tra messages. In the context of implicit discarding, R3 is executed as part of the transaction that enabled t nsition; that is, the transaction whose access to site k made UDUM1 (and hence UDUM0) detectable at th e.

3.5 Discussion

veral comments concerning the protocols and their implementation are in order.

- Each of the two protocols is composed out of a *permissive* clause and a *restrictive* clause. The permissi clause of UD/LCUM, for example, allows transactions to access both sites that are marked *locally commit* and sites that are unmarked with respect to a particular transaction. The permissive clause of LC/UDU on the other hand, allows transactions to access both sites that are marked *undone* and sites that a unmarked with respect to a particular transaction. Based on our optimistic assumptions that transacti aborts are the exception rather than the rule, it is more likely to have many locally committed markin and few undone markings. Therefore, it is likely that most of the time a typical transaction would execu at a set of sites that are either locally committed or unmarked with respect to a set of transactions, a are undone with respect to none. The dual case (where each 'locally committed' is replaced by 'undo and vica versa), is less likely to occur. Therefore, it seems that having a permissive clause based on loca committed markings (as in UD/LCUM) would result in a better protocol. A restrictive clause based locally committed marks is more likely to cause failures of the IR validation and hence transaction abor These qualitative assertions, however, must be supported by an experimental study.

- Considering the proposed implementation for both protocols, we note that the marking sets induce extra co flicts among otherwise non-conflicting pairs of transactions. The optimistic assumption favors UD/LCU in this respect, too. In LC/UDUM, otherwise non conflicting subtransactions are ordered as they execu R1 and the validity check. In UD/LCUM, on the other hand, R2 and R3 are executed only in the rare ca of a transaction abort, hence contention for the markings sets and the total order effect is diminished s nificantly. Under the optimistic assumption, most of the accesses to the marking sets in UD/LCUM wou be read accesses due to validation. For the last two reasons, it is likely that UD/LCUM will out-perfo LC/UDUM under such optimistic circumstances. However, LC/UDUM preserves the stronger IR criteri and has a very simple marker discarding mechanism.

- Deadlocks may arise due to contention to the local marking sets. For example, a transactions that read-lo *sitemarks.a* in order to perform the validation, may be blocked while attempting to access a regular da item z that is locked by CT_{ia}. The compensating transaction, on the other hand, may be blocked t holding a lock on z and attempting to access *sitemarks.a*. One simple way to avoid such deadlocks is perform all the accesses to the marking sets as the last access of subtransactions. The only problem wi this simple remedy is that late validation results in wasted efforts in case the check fails. An acceptal compromise would be to perform the check first and then unlock *sitemarks.a*. In case it succeeds and t subtransaction is completed, the validation is repeated as the last action of the subtransaction.

- Another way to reduce contention to the marking sets is to split them into individually lockable entiti one for each mark. Observe that R3 in both protocols requires locking only of the deleted mark and not t gure 7.4 illustrates this scenario. The legend for this figure is as follows: coo_2 represents the coordinator f , coo_1 represents the coordinator that initiates the marker discarding for T_1, and $DISCARD$ represents t tion of discarding the markers for T_2. An arc labeled “m” (“v”) stands for a marker carrying (validation resu essage going in the arc direction. An arc labeled “cc” (“di”) stands for a COMPENSATION-COMPLET ISCARD) message going in the arc direction. The scenario in this figure is impossible. This is realized lowing a cycle of events (1 2 3...6 1) as shown that cannot occur because the events in a distributed histo rm a partial order [Lam78].

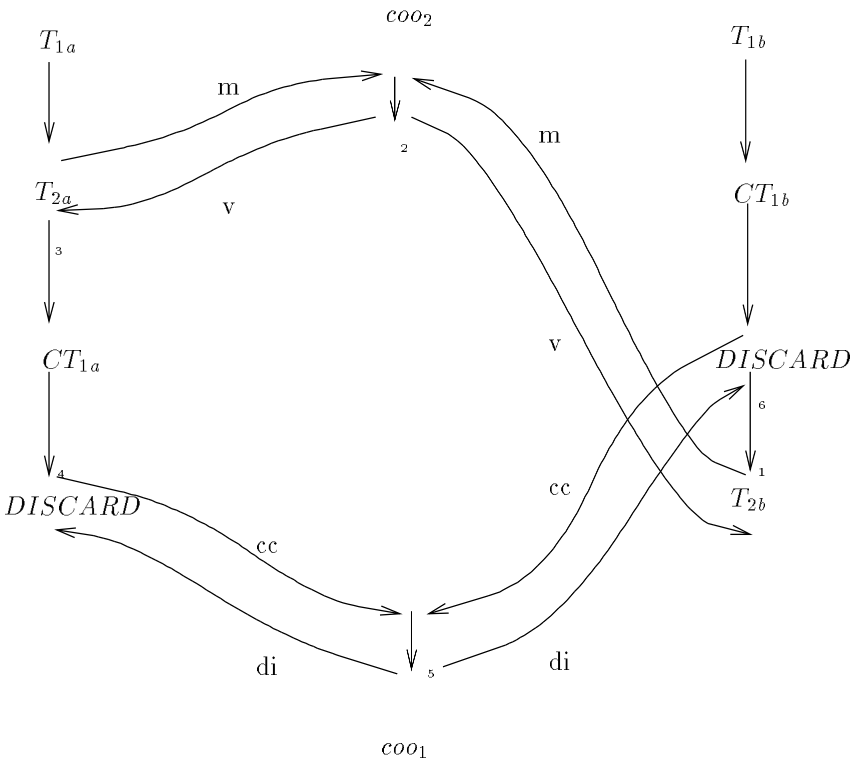

Figure 7.4: Synchronized Discarding

plicit Discarding.

use of the fact that global transactions obey the 2PL rule, the knowledge needed to detect UDUM0 can plicitly deduced rather than explicitly disseminated and gathered by extra messages. Namely, we observe th e condition in UDUM0 is implied by the following:

- *UDUM1.* For each site in which T_j executes, there is a transaction that has also executed at that site, wh that site was undone with respect to T_j.

ce a site n makes a transition in its markings as specified by UDUM1, there can be no T_j that accesses a s at was locally-committed with respect to T_j and is about to access n.

In essence, the task of synchronizing the discarding of markers is implicitly assigned to a regular transactio ther than managed explicitly by a two-phased message exchange, as in the previous method.

Lemma 14. *UDUM1 implies UDUM0.*

Proof. The proof is identical to the proof of Lemma 13, once we replace the synchronized $DISCAR$ tion of Lemma 13 by the regular transaction that implicitly synchronizes the discarding.

well as site l which is unmarked relative to T_j. Therefore, it is safe to discard the UD marker at site n a nsider it as UM. We show that enforcing the UD/LCUM rule under this circumstances assures WIR.

For the implementation of UD/LCUM, the marking of sites locally-committed with respect to transactio redundant. A binary-state marking scheme (i.e., UD and UM) suffices. As far as ensuring WIR, discardi arkers can be decoupled from the execution and commit procedure of the transactions that created the marke e caveat, however, is that presence of out-of-date markers may restrict the accesses of global transactions es, unnecessarily. On the other hand, discarding the markers too early can cause the violation of the W terion. Recall that protocol UD/LCUM allows a transaction T_s to access sites that are locally-committed wi spect to T_j as well as sites that are unmarked with respect to T_j. Therefore, T_s may access a site that is local mmitted with respect to T_j and a site that was undone with respect to T_j and was prematurely unmarked. as correctness goes, the precondition for this problematic transition is formulated as follows. A site n that done with respect to T_j can be unmarked with respect to T_j, if:

- *UDUM0 (undone to unmarked).* All T_s that have accessed sites that are locally-committed with respect T_j cannot possibly access n.

nce UDUM0 holds, the undone to unmarked with respect to T_j transition can be made safely. WIR is guarante following the UD/LCUM rule and discarding UD markers in accord with UDUM0.

The following pseudo-code segments summarize the implementation of UD/LCUM (there is no R1 in t otocol):

```
R2. The last operation of $CT_{jn}$:
$sitemarks.n \leftarrow sitemarks.n \cup \{(T_j, UD)\}$
R3. Whenever UDUM0 is detected:
$sitemarks.k \leftarrow sitemarks.k - \{(T_i, UD)\}$
```

garding R2, recall that if a site n votes to abort T_j, then the abort and the standard undo actions taken loca modeled by CT_{jn}.

Next, we present two marker-discarding techniques, both capitalizing on the opportunity to decouple t scarding from the execution of the marking transaction.

nchronized Discarding.

scarding[2] markers can be done periodically as a garbage-collection activity, thereby amortizing the associat mmunication cost over a set of transactions. Thus, it is an activity that may be relegated to light load perio riodically, a coordinator initiates a markers discarding message exchange by disseminating initiation messag a set of sites. The sites respond by including the identity of all global transactions whose local compensati btransactions completed successfully (this message is referred to as COMPENSATION-COMPLETE messag arkers can be discarded for global transactions all of whose compensations have completed as evidenced by t port from all the concerned sites. Upon receipt of this report, the coordinator sends a second round of messag the same sites, notifying them which markers can be discarded (this message is referred to as a DISCAR ssage). The two-phase message exchange creates a synchronization point which is essential for the followi etch of correctness argument.

Lemma 13. *Synchronized discarding guarantees UDUM0.*

Proof. The proof is by contradiction. Assume that there exists a transaction T_2 that accesses a site n th locally-committed with respect to T_1 and a site m after the undone marker of T_1 has been discarded at site

[2] This discarding method was designed by Nandit R. Soparkar during discussions with him on the subject. In particular, gant proof technique is due to him, and first appears in [SKS91].

R2. The last operation of CT_{jn}:
sitemarks.n $\leftarrow$ *sitemarks.n* $- \{(T_j, LC)\}$

R3. After receiving a DECISION message for T_j:
f DECISION is COMMIT **then** *sitemarks.n* $\leftarrow$ *sitemarks.n* $- \{(T_j, LC)\}$

bserve that R3 is required only to discard the LC mark and reclaim its space. It has no consequence regardi
, since T_j commits (this is a critical difference when comparing with protocol UD/LCUM).

Reasoning that protocol LC/UDUM is a correct implementation of the isolation property S1 follows from t
xt two lemmata.

Lemma 11. *If T_s accesses a site n while it is locally committed with respect T_j, then $T_{jn} \rightarrow T_{sn}$ at SG*
thout having CT_{jn} on that path.

Proof. For the IR-validation of T_s, a read access to *sitemarks.n* is generated on T_s behalf. Since t
cesses of T_{jn} and T_{sn} to *sitemarks.n* conflict, and since the history at n is serializable, T_{jn} and T_{sn} must
dered. Since T_s accesses n while it is locally committed with respect to T_j, it must be that $T_{jn} \rightarrow T_{sn}$ at SG
d CT_{jn} been on that path, the LC marker would have been removed (by R2).

Lemma 12. *If T_s accesses a site n while it is unmarked with respect T_j, T_j has executed at n not preced*
T_s, and T_j finally aborts, then $CT_{jn} \rightarrow T_{sn}$ at SG_n.

Proof. Similarly to the previous proof, since T_j, CT_j and T_s all conflict when accessing *sitemarks.n* a
ce $T_j \rightarrow CT_j$ by definition, there are two possible orders among the three transactions: $T_{jn} \rightarrow T_{sn} \rightarrow CT_{jn}$
G_x or $T_{jn} \rightarrow CT_{jn} \rightarrow T_{sn}$ at SG_x. Had the first path been a valid one, then by R1, n would have been mark
ally committed with respect to T_j.

To complete the proof of correctness of the implementation all we need to make sure is that the coordinat
forces the rule form of LC/UDUM, when it performs the IR-validation.

3.4 Protocol UD/LCUM

e main challenge in devising an implementation for UD/LCUM is the timing of the transition from undo
unmarked with respect to T_j (the arc labeled UDUM in Figure 7.3). Unfortunately, undone markers must
pt forever in order to enforce IR using UD/LCUM. To see why consider the following paths: $T_{jm} \rightarrow T_{im}$ a
$\rightarrow T_{sl}$. By the transitivity of *follows*, T_s follows T_j. IR is violated once the path $CT_{jn} \rightarrow T_{sn}$ is considere
llowing the UD/LCUM rule, T_s would be prohibited from accessing site n (thereby forming the problema
th $CT_{jn} \rightarrow T_{sn}$) at any future point, provided that the marker (j, UD) at site n is maintained forever. Th
the UDUM transition in Figure 7.3 never occurs. Discarding markers, however, is crucial for both spa
nsiderations as well as efficient execution of the IR-validation. Even in light of the optimistic assumption th
orts — and therefore UD markers — are rare, we must provide a rule for discarding UD markers for reaso
efficiency of the protocol. Interestingly, this problem does not arise in the LC/UDUM protocol.

In what follows, we describe a UD/LCUM protocol where markers are discarded, however, a weaker notion
is guaranteed. The protocol we present guarantees the following:

- *Weak Isolation of Recoveries (WIR).* No transaction is executed at a site that is locally committed wi
  respect to another transaction as well as at a site that is undone with respect to that other transaction.

IR is the incarnation of IR of Chapter 6 in the current context, since as in Chapter 6, the *follows* relation
t transitive. Observe that the above execution is WIR, since T_s accesses site n that is undone relative to

In contrast to the polarized protocol, the above protocols are based on marking sites, rather than marki
ta items. We say that a transaction accesses a site when it accesses (reads or writes) a data item residi
that site. A site is marked with respect to a transaction only if the transaction has accessed that site. T
otocols are overly restrictive since data items that are not accessed by T_i at all, and just reside in a site th
accessed by T_i are nevertheless considered as marked with respect to T_i. An improvement can be devised
arking on a data item basis and allowing propagation of markers only within a single site. Thus, discardi
arkers would have remained a local action, yet granularity of markers would have been finer.

There is a certain similarity between these protocols and the altruistic locking protocol [SGMA89]. In c
se, however, an aborted global transaction creates two *wakes* (see [SGMA89]): an undone wake and a local
mmitted wake. Similarly to the way altruistic locking restricts entering and leaving a wake, UD/LCUM a
C/UDUM restrict accessing both wakes.

In the context of a multidatabase environment, it is very important to notice that both protocols do n
pose any restrictions on local transactions. Only global transactions are subject to the restrictions posed
e protocols. Therefore, the autonomy of local database systems is not affected by these protocols.

3.2 Validating IR

e introduce data structures for maintaining the markings. For each site, n, the protocol maintains the s
emarks.n defined as follows:

$j, LC) \in sitemarks.n\ iff$ site n is locally committed with respect to T_j
$j, UD) \in sitemarks.n\ iff$ site n is undone with respect to T_j

ese *marking sets* are updated to reflect the transitions described above, and are read by global transactions
der to ascertain whether execution at a particular site complies with the relevant protocol. The fact that a s
unmarked with respect to a transaction is deduced implicitly from the lack of any marking in the correspondi
arking set. In order to preserve the semantics of the sets as defined above, concurrent accesses to the sets mu
controlled. One option is to designate special entities for storing these sets in the underlying local databas
part of the database, the sets are accessed by transactions subject to the 2PL rule. Some possible optimizatic
e discussed in Section 7.4.5.

We enforce IR in the O2PC context by using a validation method rather then by the incremental method th
s used in Chapter 6. That is, checking whether the accesses of a transaction violate the IR criterion is done
e coordinator after all the accesses have already been performed. The marking state of a site, as represented
e local *sitemarks* set, is piggy-backed with the acknowledgement/results of a completed operation. Upon recei
the markers, the coordinator validates the execution by the relevant rule (i.e., LC/UDUM, or UD/LCUM
nce the marking is on a site basis and since accesses to the marking sets are subject to the 2PL rule, sendi
e marking sets should be done only once for each subtransaction.

3.3 Protocol LC/UDUM

r the implementation of LC/UDUM, the marking of sites undone with respect to transactions is actually redu
nt, since the protocol allows transactions to access both sites that are undone and unmarked with respect
other transaction. Hence, we can simplify matters, avoid the undone marking altogether, and resort to a binar
ite marking scheme (i.e., LC and UM). The following pseudo-code segments summarize the implementati
LC/UMUD:

```
R1. After site $n$ responds to the VOTE-REQ message sent for $T_j$:
f  $n$ votes to commit $T_j$ then $sitemarks.n \leftarrow sitemarks.n \cup \{(T_i, LC)\}$
```

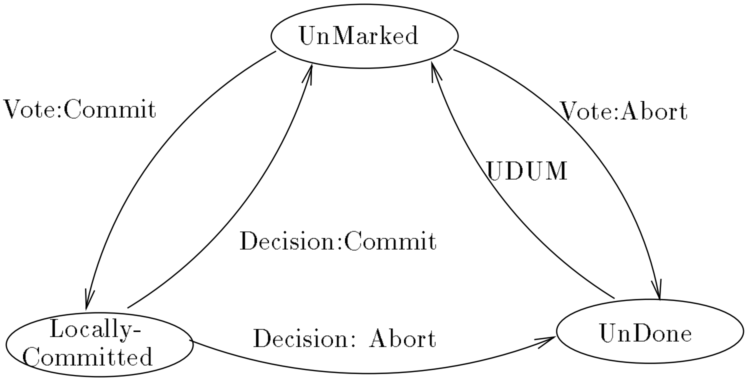

Figure 7.3: Transitions in the marking of a site

3.1 Marking Sites

e basic building block for implementing protocols that are based on the isolation properties is a simple marki sites. With respect to a specific global transaction T_j, a site is either *unmarked (UM)*, or *marked*. Then, a si marked *locally-committed (LC)* with respect to T_j, or marked *undone (UD)* with respect to T_j. Initially, a si unmarked with respect to a transaction T_j. A site is made locally-committed with respect to T_j once it vo commit T_j in response to a VOTE-REQ message. On the other hand, if the site votes to abort T_j, the site ade undone with respect to T_j. A site ceases to be locally-committed with respect to T_i and becomes unmark th respect to that transaction whenever the site receives the decision message from the 2PC coordinator mmit T_i. If the decision is to abort T_j, then the site becomes undone with respect to T_j. At some poi site ceases being undone with respect to an aborted transaction and becomes unmarked with respect to th nsaction. We postpone the discussion concerning this transition to Section 6.3.2. It is important to note th these transitions in the marking are triggered either by local events, or by messages that are already part e 2PC protocol. Figure 7.3 summarizes the transitions in the markings.

Using this marking scheme, we devise protocols that ensure that the isolation properties are satisfied. Int vely, the protocol should prevent situations where a global transaction accesses a site that is locally-committ th respect to another transaction, as well as a site that is undone with respect to that other transaction, sin ch a situation can result in a non IR history. Protocols LC/UDUM and UD/LCUM correspond to the isolati operties S1 and S2, respectively. Each of the two protocols can be summarized by a rule that restricts the sit global transaction T_s may access:

- *LC/UDUM.* Let T_s execute at a site that is marked with respect to a T_j. Then for each such T_j, either o of the following conditions hold:
  - all sites in which T_s executes are locally-committed with respect to T_j.
  - all sites in which T_s executes are either undone or unmarked with respect to T_j.
- *UD/LCUM.* Let T_s execute at a site that is marked with respect to a T_j. Then for each such T_j, either o of the following conditions hold:
  - all sites in which T_s executes are undone with respect to T_j.
  - all sites in which T_s executes are either locally-committed or unmarked with respect to T_j.

Starting with the premise that T_s follows CT_{jc}, and using a symmetric argument, C2 is similarly proven.

Lemma 9 is pictorially illustrated in figure 7.2 which describes both the global and local SGs. The figur rrespond to a history where the second disjuncts in the second conjuncts of C1 and C2 hold.

Next, we introduce two properties of global SGs that are used to 'isolate' non-atomic executions, there suring IR. Each property is presented as a formal assertion. We first introduce four predicates that depend e transaction identifiers j and s:

- *A1(j,s)*: At any SG_a where T_s appears, $T_{ja} \rightarrow CT_{ja} \rightarrow T_{sa}$.
- *A2(j,s)*: At any SG_a where T_s appears, $T_{ja} \rightarrow T_{sa}$ without having CT_{ja} on that path.
- *A3(j,s)*: At any SG_a where both T_s and T_j appear, if there is a local path $T_{ja} \rightarrow T_{sa}$, then the pa $T_{ja} \rightarrow CT_{ja} \rightarrow T_{sa}$ is in SG_a.
- *A4(j,s)*: At any SG_a where both T_s and T_j appear, if there is a local path $T_{ja} \rightarrow T_{sa}$, then the pa $T_{ja} \rightarrow T_{sa}$ is in SG_a, without having CT_{ja} on that path.

Using these predicates we introduce two *isolation properties*:

- *S1*: $(\forall T_j, T_s : T_j \rightarrow T_s$ in the global SG $: A2 \vee A3)$
- *S2*: $(\forall T_j, T_s : T_j \rightarrow T_s$ in the global SG $: A1 \vee A4)$

Lemma 10. *The following assertions hold:*

- $(\exists T_j : C1(j)) \Rightarrow \neg S1$*; and*
- $(\exists T_j : C2(j)) \Rightarrow \neg S2$*.*

Proof. Consider the path $CT_{jc} \rightarrow T_{kc}$ in SG_c whose existence is guaranteed by the first conjunct of C cause of this path and since CT_{jc} is always serialized after T_{jc}, we have that $T_j \rightarrow T_k$ in the global SG. By t cond conjunct of C2, there exists an SG_d where either $T_{jd} \rightarrow T_{kd}$ without having CT_{jd} on that path, or the no path between T_j and T_k in SG_d. In both cases, the negation of $A1(j,k)$ is implied. Considering the pa $T_{jc} \rightarrow T_{kc}$ in SG_c again, we observe that the negation of $A4(j,k)$ holds. Therefore, we have demonstrated th T_j and T_k, where $T_j \rightarrow T_k$ in the global SG, both $\neg A1$ and $\neg A4$ hold.

By a symmetric argument the second part of the lemma follows.

Theorem 6. *If either one of the isolation properties S1 or S2 hold, then the execution is IR.*

Proof. Let i be either 1 or 2, then:

$$\begin{array}{ll} & \text{The history is not IR} \\ \Rightarrow & \{\text{ Lemma 9 }\} \\ & (\exists T_j : C1(j) \wedge C2(j)) \\ \Rightarrow & \{\text{ weakening }\} \\ & (\exists T_j : C1(j)) \wedge (\exists T_j : C2(j)) \\ \Rightarrow & \{\text{ Lemma 10 }\} \\ & \neg S_i \end{array}$$

he counter positive form of this implication is the theorem statement.

.3 Protocols for Isolation of Recoveries

this section, we present two protocols that ensure IR when the O2PC protocol is employed. As such, t otocols actually complement the O2PC protocol. The protocols implement the isolation properties. We stri protocols whose execution requires no messages other than the standard 2PC messages.

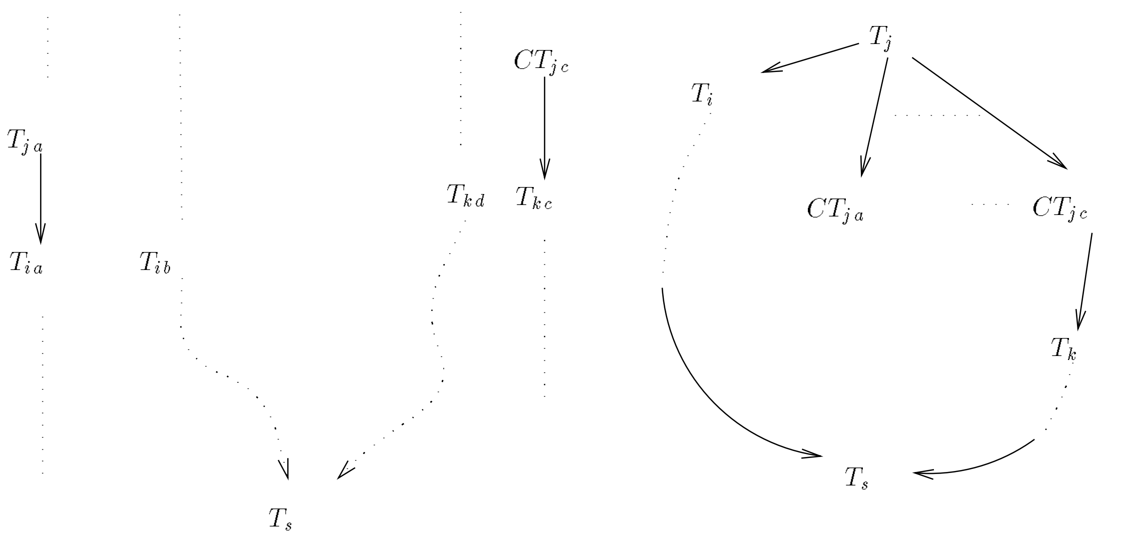

Figure 7.2: Illustrating Lemma 9 by the local SGs (left) and global SG (right)

- A transaction T_i *follows* a forward transaction T_j in a history, if $T_j \rightarrow T_i$ is a path in the corresponding S and there is no compensating subtransaction CT_{jn} on that path.

llowing a compensating subtransaction is transitive. Following a forward transaction is transitive, except wh orresponding compensating subtransaction appears in the path.

Lemma 9. *If an execution under O2PC is not IR, then there exists a global transaction T_j such that:*

- *C1(j): There exists a global transaction T_i $(i \neq j)$ such that $T_{ja} \rightarrow T_{ia}$ at some SG_a, without having CT on that path, and at some other SG_b where T_i appears, either $CT_{jb} \rightarrow T_{ib}$, or there is no local path betwe T_j and T_i in SG_b; and*
- *C2(j): There exists a global transaction T_k $(k \neq j)$ such that $CT_{jc} \rightarrow T_{kc}$ at some SG_c, and at some oth SG_d where T_k appears, either $T_{jd} \rightarrow T_{kd}$ without having CT_{jd} on that path, or there is no local path betwe T_j and T_k in SG_d.*

Proof. For the purpose of the following proof, we need to define shortest path between two transaction noc a global SG. Let us segment paths in the global SG into local paths. The *shortest path* between two transacti des in the global SG, is the global path connecting these two nodes with the least number of segments (t ortest path may not be unique).

Since the execution is not IR, there exist T_s and T_j, such that T_s follows T_j, as well as T_s follows CT_{jn} f me site n. Consider the shortest path $T_j \rightarrow T_s$, where there is no compensating subtransaction CT_{jm} on th th. Such a path exists by the definition of T_s follows T_j. Let the first segment of this path be in SG_a, a e second in SG_b. Furthermore, let T_i be the last global transaction on the first segment and the first one e second segment (see the left figure in Figure 7.2 for illustration). We have that $T_{ja} \rightarrow T_{ia}$, without havi T_{ja} on that path, thereby satisfying the first conjunct of C1. Consider the next site b on that shortest pat $CT_{jb} \rightarrow T_{ib}$, then the first disjunct in the second conjunct of C1 holds. If such a path does not exist, th claim that there is no path between T_j and T_i in SG_b, and hence the second disjunct of the conjunct of lds. First, by Lemma 8, it cannot be that $T_{ib} \rightarrow T_{jb}$, since a cycle would have formed in the global SG (sin ${}_a \rightarrow T_{ia}$ in SG_a). Second, had $T_{jb} \rightarrow T_{ib}$, the path $T_j \rightarrow T_s$ with a first segment at SG_a would not have be e shortest path between T_j and T_s.

- We give a formal definition of IR in the context of O2PC.

- We show that if an execution is *not* IR then certain conditions are implied (Lemma 9).

- We introduce properties of SGs, called *isolation properties* whose negation is implied by the above conditic (Lemma 10).

- We conclude in Theorem 6 that by ensuring the isolation properties, IR histories are guaranteed.

As pointed out in Section 5.6, there are differences in the underlying transaction models used in Chapters d 7. Consequently, there are distinctions in the presentation of the IR criterion, even though the basic notion entical. In this chapter, the *follows* relation is defined in terms of paths in serialization graphs (SGs) (which a nilar to the partial orders introduced by composite executions in Chapter 6). Our SGs are a slightly extend rsion of the traditional SGs, since they include nodes for subtransactions that aborted during the execution e O2PC protocol. Inclusion of these subtransactions in the SGs is crucial for the definition of IR. As in t ndard treatment of SGs, subtransactions that are aborted earlier are not accounted for in the SGs. To ma e presentation uniform, we use the syntactic device of modeling a subtransaction that aborts during the O2I otocol as a committed subtransaction followed immediately by the corresponding compensating subtransactic ctually, an abort followed by a standard roll-back is a special case of compensation, where no transaction ha ad from the compensated-for transaction [KLS90a]). Using this syntactic transformation, we need not u larities to define IR. Following are the formal definitions and results.

Let $\mathcal{T}$ be a set of global transactions, and let $\mathcal{CT}_a$ be the set of the corresponding compensating subtrar tions at site a. The *local serialization graph* for site a for a complete local history[1] H_a is a directed gra $G_a(H_a)$=(V_a, E_a). The set of nodes V_a consists of a subset of transactions in $\mathcal{T} \cup \mathcal{CT}_a$. The set of edges nsists of all $A \rightarrow B$, $A, B \in \mathcal{T} \cup \mathcal{CT}_a$, such that one of A's operations precedes and conflicts with one of I erations in H_a.

A *global SG* is an SG that corresponds to a history at more than one site. Given a set of local SGs, ea presented as $SG_a = (V_a, E_a)$, the corresponding global SG is defined as $SG_{global} = (\cup\, V_a,\, \cup\, E_a)$. Observe th ch compensating subtransaction is assigned a separate node in the global SG (in accord with the localizati compensation principle).

Lemma 8. *A global SG that corresponds to a history under the O2PC protocol is acyclic.*

Proof. The O2PC protocol assumes that local histories are serializable, and hence local SGs are acycl ie presence of compensating subtransactions cannot introduce cycles, since each compensating subtransaction presented as a separate node. As was already mentioned, the O2PC protocol preserves synchronization poin d hence each global transaction still follows the 2PL rule, globally. Therefore, the global SG is acyclic.

Before we proceed, we establish some notation. The notation $A \rightarrow B$ is used to denote that there is a direct th (of arbitrary length) between the two transaction nodes in a given SG. When specifying a local path, the lo it belongs to, is also specified. A global transaction T_i that requires access to data located at sites $1, 2, \ldots$ submitted for execution as a collection of local subtransactions $T_{i1}, T_{i2}, \ldots, T_{ik}$, where T_{ij} is executed at site nilarly, CT_{ij} is the compensating subtransaction at site j that corresponds to the forward subtransaction T

In the definition of *follows*, we distinguish between following a compensating subtransaction, and followi orward transaction:

- A transaction T_i *follows* a compensating subtransaction CT_{jn} in a history, if $CT_{jn} \rightarrow T_i$ is a path in t corresponding SG.

[1] See [BHG87] for precise definitions of complete histories.

Under certain circumstances, the O2PC scheme can be employed as it was presented so far, without a
rther adjustments. If transactions are not sensitive, and hence the notion of IR is not relevant for them, O2I
n be employed right away.

Another simple way of taking advantage of the O2PC idea without tackling correctness issues is to allow *o*
al transactions to benefit from the fact that global transactions release their locks early. That is, a glob
nsaction releases its locks and becomes locally-committed only for the purposes of letting local transactio
oceed; other global transactions are still delayed. This simple version of the O2PC protocol reduces the leng
time local transactions are delayed due to global transactions.

.1 O2PC in Real-Time DTM Systems

this section we briefly mention several relevant aspects of the work reported in [SLKS91], where compensati
used in the context of a real-time DTM system.

The harsh consequences of enforcing atomicity in DTM systems cannot be tolerated in typical real-ti
plications. Under light system loads and no failures, using 2PC is acceptable. However, when those assumptio
not hold, an alternative is needed. An adaptive approach is taken in [SLKS91] that permits a site to dynamica
itch to the less costly O2PC under situations that demand it, such as a transient excessive load. The decisi
switch between the two commit protocols can be taken autonomously at any site. Switching between t
otocols exploits a trade-off between the cost of commitment and the obtained degree of atomicity. Name
v-cost protocol and relaxed atomicity under O2PC, and high-cost protocol and standard atomicity under 2P

As was already pointed out, there is more overhead to compensation than standard recovery. The feature
mpensation that is crucial to its applicability to real-time systems is that undo operations must be perform
mediately, while compensatory action may be deferred. This allows recovery work to be performed duri
riods of light system load despite the expectation that transaction failures (and thus recovery) will occ
sproportionately more during times of high system load. Furthermore, it is not necessary for a failed transacti
to hold data pending the execution of CT. Rather, a failed T can release data that is later (re)acquired
T. Since we allow standard 2PC to be used as the norm for transaction commitment, with compensation-bas
chniques invoked only when time-constraints require it, the overhead is further reduced.

To substantiate the above claims, an example adapted from the real-time systems literature was worked ou
e example is concerned with a tracking system for mobile targets [Son88, Koo90]. The system is dispers
er several processing sites that manage target-sensors, target-tracking weapons, and store data pertaining
e readings, positions, etc. in local database systems. Periodically, the sensors update the data regarding t
gets as sensed at each local site, and this data is also sent to a specific coordinator site. The coordinator si
ceives such track data from several sites and correlates the information gathered to create the global tracki
formation. In the example, compensation is used to amend the positioning of the weapon system after
oneous reading is recorded at one of the stations (say, due to a signal-processing error). Compensation
rformed by positioning the weapon system based on extrapolation of its past trajectory since the local site do
t know the precise correct current position for it.

.2 Isolation of Recoveries under the O2PC Protocol

e main result of this section is the derivation of a sufficient condition for obtaining the IR criterion under t
PC protocol. The strategy in obtaining the main result is summarized as follows:

Section 7.4.

e one hand, the early release of locks solves the problems of blocking and the local commitment keeps t es autonomous. On the other hand, the uncoordinated commitment of updates may violate the standa -or-nothing atomicity guarantee of a transaction, if at least one of the sites votes to abort it.

As was outlined above, we use compensating transactions, in conjunction with the O2PC protocol, as t eans to ensure transaction atomicity despite of the uncoordinated commitment of updates at different sit ter voting to commit T, a site still carries on with the second phase of the regular 2PC protocol (despite havi eased locks held by T). If the site receives a decision message from the coordinator to abort T, then it invol e corresponding compensating transaction. Since it is more likely that the decision would be to commit T, t in by the early release of locks should outweigh the overhead associated with those cases requiring compensati T. The message transfer in the O2PC protocol between the coordinator and a participating site is depict ctorially in Figure 7.1.

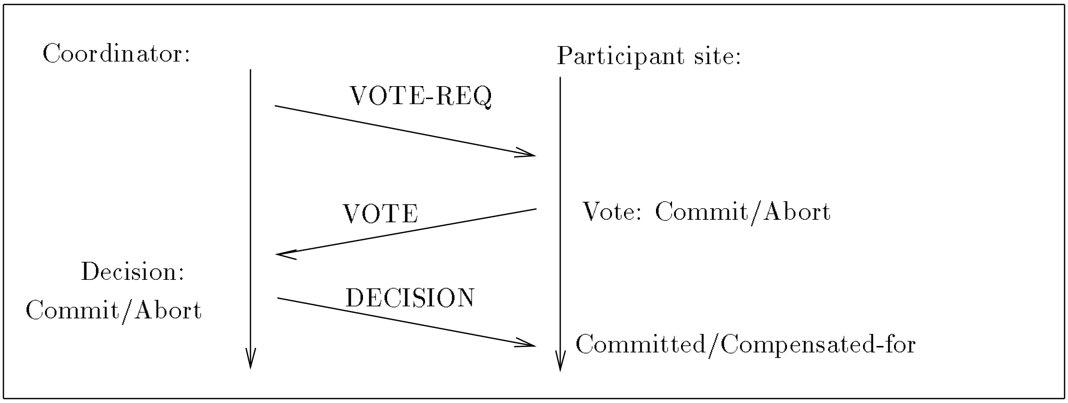

Figure 7.1: A schematic view of the O2PC protocol

The execution of a compensating transaction requires access to the log and other information stored on stal orage (see Section 4.1), thus further increasing the cost associated with this type of transaction. For the asons, we limit the usage of compensating transactions in our context, for relatively rare pessimistic cases lures of global transactions.

In the context of the O2PC protocol, compensation is employed as follows. If T_j is a global transactio $T_{i2}, \ldots, CT_{ik}$ are local *compensating subtransactions*, one for each site where T_i was executed. Each compensa g subtransaction is submitted for execution at a site just like any other local transaction, and hence it is subje the local concurrency control. Compensating subtransactions are treated as local transactions rather than btransactions of global transactions with respect to locking; that is, they also follow strict 2PL locally. Ther e, at each site, the local execution over local transactions, subtransactions, and compensating subtransactio serializable.

A distinctive feature of the O2PC protocol is that it makes no changes to the message transfer pattern e structure of the standard 2PC protocol. Even when O2PC is augmented to preserve IR (in Section 7 e structure of 2PC is preserved. The changes are in local reactions to the 2PC messages. Therefore, O2I es not contradict standardization efforts of the 2PC protocol. Moreover, there is a very strong compatibil tween 2PC and its optimistic variant. Transactions employing the former can be executed concurrently wi nsactions obeying the latter, and still global transactions follow 2PL globally. This guarantee follows fro e fact that O2PC preserves the synchronization points of subtransactions. Furthermore, even for the sar obal transaction, some of the constituent subtransactions may be engaged in O2PC and some in 2PC. The vantageous properties are exploited in the work reported in Section 7.2 [SLKS91], and in the ideas describ

Chapter 7

The O2PC Protocol

different method for achieving relaxed atomicity in a DTM environment is presented in this chapter. T ethod is based on the *optimistic 2PC (O2PC)* protocol. Formal definition of the IR criterion in this context ven in Section 7.3 and corresponding protocols are given in Section 7.4.

In the standard 2PC protocol, a multi-site transaction is associated with a coordinator that initiates t otocol by sending a VOTE-REQ message (also referred to as PREPARE message) to all participating sit oon receipt of this message, a participating site votes (by sending a VOTE message back to the coordinat her to commit the particular transaction or to abort it. Based on these votes, the coordinator decides wheth commit or abort the transaction. Only if all the votes are to commit then the transaction is to be committ llowing this, the coordinator transmits its DECISION message to the participating sites.

Typically, in DTM systems global serializability is obtained using the synchronization points techniques ibed in Section 5.4.3 that combines the 2PC protocol with a global 2PL discipline. For the well known reaso avoiding cascading aborts, and use of state-based recovery, the exclusive (i.e., write) locks are released o er the DECISION message is received locally. Thus, a *strict* version of 2PL is used. It is possible to relea shared (i.e., read) locks as soon as the VOTE-REQ message is received.

Holding the locks until a DECISION message is received, which is the cause of blocking, is necessary o the transaction at hand has to be aborted. Our revised protocol is based on the *optimistic assumption* th most cases the protocol terminates successfully (i.e., the transaction commits) and therefore the locks can eased earlier. This can dramatically reduce waiting due to data contention, thereby improving the performa the system. Such an assumption is valid in most practical distributed environments. Furthermore, since t mmit protocol is initiated only when the transaction at hand has already obtained all its locks and complet its operations, its failure is very unlikely. Namely, the transaction cannot participate in a deadlock, nor c fail because if a logical error. It can fail only because of site or communication faults, which usually are rath frequent. The validity of the optimistic assumption is orthogonal to the protocol correctness. However, if t sumption is unfounded, the overhead incurred by the protocol is likely to outweight its benefits.

The optimistic 2PC (O2PC) protocol is a slightly modified version of the 2PC protocol. The same messa change is carried out as in the standard protocol. If a site votes to abort T_j, then as in the standard protoc abort vote is sent back to the coordinator, and the locks held by the transaction are released as soon as t nsaction is locally undone (rolled-back). However, if a site votes to commit T_j, *all locks held by T_j are relea once, without waiting for the coordinator's final commit or abort message.* In this case, we say that T_j *ally-committed* at that particular site. Observe that a global 2PL discipline is preserved, even under the ea k release provision of the O2PC protocol.

The uncoordinated local commitment resulting in the early release of locks is the crux of the protocol. (

follows for a more conservative transaction model is given in Chapter 7. There, in spite of the transitivi opagation of effects is controlled by dealing with sites as the unit of marking.

Another question that should arise concerns an execution where T_{32} is serialized in between T_{12} and T_{22} gure 6.1 (b). By the definition, this execution turns out to be IR. (This execution is not IR, however, under t finition of *follows* given in Chapter 7). This should not be surprising once considering the following reasonir e made a modeling decision that if such a T_{32} propagates the effects of T_{12} to T_{22}, then it should have beer btransaction of T_1 itself, and not a subtransaction of a different transaction. That is, our transactions crea heres of atomicity and consistency. All subtransactions that are related to a single activity in terms of causali omicity and consistency should be grouped as a single composite transaction [tm-91, GMGK$^+$90]. If T_3 is parate transaction, then the sensitivity of T_2 is not an issue any more, once T_{32} executed in between T_{12} a $_2$.

.5 When Actions Are Not Semantically-Recoverable

ne concept of relaxed atomicity relies on the methods of compensation and retry. As was mentioned earli ese methods are not applicable universally, and are based on semantics of the applications at hand. For instan insactions involving *real actions* [Gra81] (e.g., firing a missile or dispensing cash) may not be compensatab ne adjustment for transactions involving non-compensatable subtransactions is to retain their locks and del al actions until a commit decision message is received from the coordinator (as in standard two-phase comm all sites performing these actions. All other sites running compensatable subtransactions on behalf of the sa insaction can still benefit from the early lock release of our modified commit protocol.

A general way of integrating arbitrary subtransactions (which may not be suitable for compensation or retr o our model, is described next. Each subtransaction can be divided into three portions: a compensatab rtion (CP), a pivot portion (PP), and a retriable portion (RP). The execution of such a subtransaction wou oceed as follows: The CP is executed first, and following its termination all locks it has acquired are releas once. The PP is executed second and its termination is coordinated by a 2PC protocol among all the pivo the subtransactions of the same transaction. While waiting for the 2PC decision message to arrive, the I initiated. Locks acquired by the PP or the RP are released only after a decision message is received local the decision is to abort, then the RP is aborted and both RP and PP are undone using standard recove ice their locks have not been released. Additionally, the CP is compensated-for. If, however, the decision commit, then PP's locks are released, and if RP happens to have failed it is re-executed until it succeeds. sadvantage of this method is that because of the very early release of locks by the CP, synchronization poi e not preserved, and thus 2PL property of the global transaction is lost.

ction 2.1 is used and hence executions are eventually semantically atomic. The following propositions assu semantically atomic execution E, that is generated under the polarized protocol. The lemmata are dire nsequence of the rules used to present the protocol.

Lemma 3. *If q_j follows s_i^b because of a conflict on x at a certain site, and at that site T_i's markers ha t been discarded by the time q_j accesses x, then $(i,b) \in mfb(q_j)$.*

Lemma 4. *If $q_j \xrightarrow{j} p_j$, then $mfb(q_j) \subseteq mfb(p_j)$.*

Lemma 5. *A marker (i,b) is discarded at site, only if there is no active transaction T_j such that $(i,\bar{b})$ fb(o_j) at any other site.*

Lemma 6. *All markers in the follow set of a subtransaction are of the same polarity.*

Lemma 7. *If o_j follows p_i^b because of a conflict on a data item, and T_i's fate is $\bar{b}$, then $(i,b) \in mfb(o_j)$.*

Proof. Let o_j and p_i conflict on x. Since T_i has a unanimous fate, a recovery subtransaction that correspon p_i must exist in the execution. Thus, the marker $(i,b) \in access(x)$ is discarded by the recovery subtransacti at corresponds to p_i. Therefore, since o_j follows p_i^b, o_j must precede this recovery subtransaction, and hen e marker would not be removed prior to o_j's access to x. Consequently, $(i,b) \in mfb(o_j)$.

Theorem 5. *The polarized protocol ensures that executions isolate recoveries.*

Proof. The proof is by contradiction. We assume that p_j follows s_i^b and $t_i^{\bar{b}}$, and derive a contradiction. I e definition of the 'follows' relation, there are subtransactions q_j (o_j) that follow s_i^b ($t_i^{\bar{b}}$) because of confli data items x (y) at site 1 (site 2), and $q_j \xrightarrow{j} p_j$ ($o_j \xrightarrow{j} p_j$). (One of q_j, o_j may be p_j itself). By assumptio e execution is SA, and hence T_i has a unanimous fate b or $\bar{b}$. Without loss of generality it may be assum at the final fate of T_i is b (the dual case is symmetric). By Lemma 7, $(i,\bar{b}) \in mfb(o_j)$. The proof proceeds nsidering two cases. First, we assume that site 1 had already discarded the marker (i,b) when q_j accessed x. I mma 5, the (i,b) marker could have been discarded only if $(i,\bar{b}) \notin mfb(o_j)$ at site 2. However, $(i,\bar{b}) \in mfb(o$ contradiction is derived. In the second case, we assume that the marker (i,b) was not discarded before cessed x at site S_1. Then, by Lemma 3, $(i,b) \in mfb(q_j)$, and by Lemma 4, $(i,b) \in mfb(p_j)$. However, sin $\bar{b}) \in mfb(o_j)$, by Lemma 4, $(i,\bar{b}) \in mfb(p_j)$, too. This contradicts Lemma 6.

.4 Discussion

few comments concerning several modeling decisions that have been made in this chapter are in order. T ation *follows* is not transitive, and this is a critical point. Intuitively, this relation models the propagati the effects of a subtransaction t_j on subtransactions of other transactions. Such propagation is allowed or thin a single transaction, one of whose subtransactions came immediately after t_j. The reasoning behind t cision to limit this propagation is that we wanted to confine the cascading effects of a subtransaction someho sume that a subtransaction p_j commits, and is followed by other subtransactions transitively. If T_j is abort need to compensate for p_j, and also we need to compensate for the uncontrollable cascading effects of is very much resembles cascading aborts, which the method of compensation is set to prevent. In particul nsider the following $<_E$ orderings: $t_j <_E q_i$ where both subtransactions execute at the same site, and $q_i <_E$ here p_k executes at a different site. Had *follows* been transitive, p_k would have followed t_j. Such uncontrollal opagation to remote sites is troublesome.

Propagation of effects is modeled and tracked by the propagation of markers in the protocol. Making follo nsitive means that a marker (i,b) would have to be assigned to $access(x)$ even if p_i does not access x. problematic to discard such arbitrarily scattered markers. Currently we maintain the invariant that (i,b) $cess(x)$ only if a subtransaction of T_i accesses x. This invariant enables to discard markers of a particul nsaction by local actions at all sites where the transaction was executed. A different, transitive, definiti necessarily. Therefore, it is necessary to discard markers.

Discarding markers should be done carefully, since discarding a marker too early can lead to incorrectly passi e condition of the second and third rules, thereby generating a non-IR execution. Such an undesirable scenar n arise if a subtransaction p_j follows q_i^b, and accesses a data item whose $(i, \bar{b})$ marker was removed premature far as correctness goes, the precondition for discarding markers is formulated as follows.

> A marker (i, b) of a transaction T_i (with a unanimous fate), can be discarded if all T_j whose subtransactions follow $q_i^{\bar{b}}$ are no longer active.

A transaction is active until it can no longer initiate new subtransactions at new sites (i.e., until the coordinat tiates the commit protocol for that transaction). Once T_j becomes inactive, subtransactions p_j can no long use the problem outlined above. Implementing this transition requires cooperation among sites and hence ext mmunication. It should be emphasized, however, that this additional message exchange is needed only for t rposes of discarding markers, and it can be decoupled from the execution and commit procedure of a particul nsaction. Specifically, discarding markers can be done periodically, as a garbage collection activity, there nortizing the communication cost over a set of transactions.

The purpose of the following message exchange is to discard markers of transactions executed at a set rticipating sites and coordinated by a coordinator. This message exchange is executed periodically, and not f ery transaction separately. The additional rules for this exchange are described next.

- **At a participant** as a response for a request to DISCARD from the coordinator. For transactions whose local recovery subtransactions have terminated successfully with polarity b:

```
if ¬(∃ local subtransaction $p_j : (i, \bar{b}) \in mfb(p_j)$)
then send SAFE ack message to coordinator
else send UNSAFE ack message to coordinator
```

- **At the coordinator.** When the coordinator has received SAF UNSAFE acks from all participants of T_i that executed recovery subtransactions for T_i:

```
if all the participants sent a SAFE ack
then notify all participants to discard $T_i$'s markers
```

Having received all the SAFE/UNSAFE acks for a transaction T_i, the coordinator has all the informati eded to determine whether it is safe to discard T_i's markers. First, the coordinator is certain that T_i has animous fate, since the successful termination of all of T_i's recovery subtransactions was acknowledged. It is sa discard T_i's markers provided that there is no transaction T_j, whose subtransaction follows a subtransacti T_i with a polarity opposite to the final fate of T_i. The existence or absence of such a transaction T_j is encod the SAFE/UNSAFE ack messages.

It is assumed that mfb sets of subtransactions are maintained as long as the corresponding transaction tive. Finally, it should be noted that discarding markers at a site need not be performed as a synchrono tion. By the time a site is notified that it can discard markers, their presence or absence is of no consequen atsoever.

3.3 Correctness

e are in position now to state several results concerning the protocol. The polarized protocol is a reacti gorithm that reacts to events in the course of processing a multi-site transaction. The execution of transactio governed by the protocol, and hence only a certain class of executions is allowed under the polarized protoc ir objective is to show that these executions isolate recoveries. We assume that the commit protocol outlined

- For each subtransaction p_j the protocol maintains the following set:
  $mfb(p_j)$, the set of all markers (i, b) such that p_j follows[2] q_i^b. (The name $mfb(p_i)$ should be read as "mark
  followed by p_i").

tially, for all data items x, and for all subtransactions p_j, $access(x) = \emptyset$ and $mfb(p_j) = \emptyset$, respectively.

Regarding the first rule below, the set subtraction is effective only for a successful recovery subtransacti at removes the marker of its corresponding forward subtransaction. The second rule propagates markers on cases of conflicts among subtransactions. Dependency orderings may not be based on data conflicts, but st ke part in 'follows' chains. The third rule takes care to reflect these dependency orderings in the *mfb* se nce dependency orderings are inter-site, this rule implicitly assumes the communication needed for the remo vocation.

1. **Marking.** Whenever a forward or a recovery subtransaction s_i terminates with polarity b, then for all da
   items x accessed by s_i:
   $access(x) := (access(x) \; - \; \{(i, \bar{b})\}) \; \cup \; \{(i, b)\}$

2. **Access and Propagation.** Whenever a subtransaction p_i requests access to data item x:
   if $(\exists j \, : \, (j, b) \in mfb(p_i) \; \wedge \; (j, \bar{b}) \in access(x))$
   then reject p_i's request
   else $mfb(p_i) := mfb(p_i) \cup access(x)$

3. **Propagation by Dependence.** Whenever a subtransaction p_i requests to invoke a subtransaction q_i:
   if $(\exists j \, : \, (j, b) \in mfb(p_i) \; \wedge \; (j, \bar{b}) \in mfb(q_i))$
   then reject p_i's request
   else $mfb(q_i) := mfb(q_i) \cup mfb(p_i)$

Executions that are not IR are excluded by checking the conditions of the second rule and the third rule. btransaction is prevented from accessing data items marked by markers of the same transaction with opposi larity. If a subtransaction attempts to access a data item x that would violate this condition, the access reque rejected in the second rule. Rejection implies that the subtransaction can either fail or be delayed. Delaying eful only if it is possible that x's offending marker will be removed. Such a removal may occur when a success covery subtransaction replaces the offending marker with the opposite marker as prescribed in the first ru ependency orderings that violate the IR criterion are similarly rejected in the third rule.

Recall that invoking a subtransaction may model initiating it, as well as any other type of dependency betwe btransactions (e.g., data flow, synchronization). The only time where $mfb(q_i)$ is not empty in rule 3 is when tw btransactions "invoke" the same third subtransaction within the same transaction. Such an invocation mod synchronization event, or a subtransaction that gets input from the output of two previous subtransactions. s case, $mfb(q_i)$ accumulates markers and rule 3 is executed as a validation check (i.e., after the subtransacti is already active and got its inputs for example).

Observe that the rules of the protocols check local conditions and prescribe local actions, and hence, a lo heduler can implement the protocol without additional inter-site communication.

3.2 Discarding Markers

r the condition checking in rules 2 and 3 to be performed fast and for space efficiency, it is required to ke e $access(x)$ sets finite. Moreover, if markers are not discarded in a timely fashion, they restrict access

[2] Since discarding markers is done only periodically and asynchronously (see below), the mfb sets may accumulate extra marke at is, it may indeed happen that (i, b) is in $mfb(p_j)$ and p_j does not follow a q_i. Such a situation arises because of Rule 2 wh igns to $mfb(p_i)$ regardless of the non-transitive definition of follows.

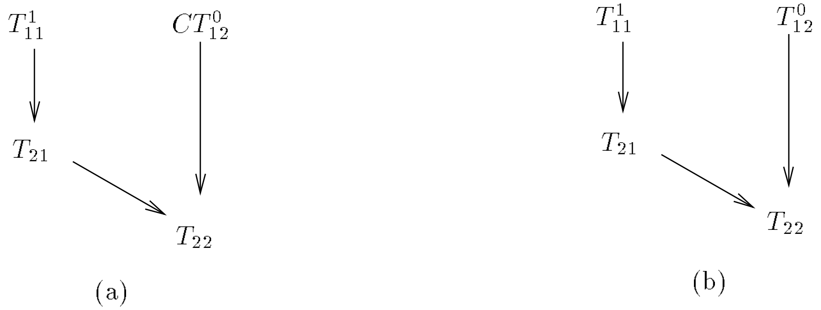

Figure 6.1: Non-IR executions

In an IR execution it is possible to follow a forward subtransaction, p before a successful rp actually revers effects. The class of IR executions excludes, however, executions in which a global state resulting from complete execution of a transaction is observed by other transactions. Thus, a notion of virtual atomicity forced. A nice feature of the definition of IR using polarities, is that it excludes following all possible non-aton mbinations among committed, aborted, compensated-for, and retried subtransactions.

To illustrate, consider the sample executions depicted in Figure 6.1. In this figure a subtransaction of transa on T_i executing at site S_j is denoted T_{ij}. The notation CT_{ij} denotes the compensating subtransaction speci the forward subtransaction T_{ij}. The $<_E$ relation is depicted by arrows. In this figure all the executions t IR.

The example in Figure 6.1 (a) is of particular interest. By following CT_{12}, T_{22} "knows" that the coordinat T_1 has decided to abort T_1. On the other hand, the effects of T_{11} are visible at site S_1, and thus affect T_{21} a nsitively T_{22}. T_{22} is exposed to the asynchrony in the process of recovering T_1, and consequently may obser inconsistent state.

.3 The Polarized Protocol

this section, we present a protocol, called the *polarized protocol*, that ensures that executions isolate recoveri e protocol is executed by schedulers at each site that collectively ensure the IR property. The protoc plements the 'follows' relation which is crucial to the generation of IR executions. It does so by *marking* da ms with polarities of subtransactions that access them, and *propagating* these markings along conflict a pendency orderings.

First, we present the general polarized protocol which applies regardless of the type of decision rule t ordinator employs. The protocol, expressed as a set of rules, and some explanatory comments are included ction 6.3.1. Section 6.3.2 focuses on the problem of discarding markers. A correctness proof is presented ction 6.3.3.

3.1 The Protocol

e following type and data structures are used in the protocol:

- A *marker* is an ordered pair (i, b), where i is a transaction identifier and b is a polarity of a subtransacti within that transaction.
- For each data item x the protocol maintains the following set:
  $access(x)$, the set of all markers (i, b) such that x is accessed by a subtransaction p_i^b.

The *fate* of a transaction T_i in an execution E is the union of the set of the polarities of T_i's recove btransactions in E, with the set of polarities of subtransactions of T_i that have no recovery subtransactio rmally:

$$fate(T_i, E) = \{b \mid rp_i^b \in E\} \cup \{b \mid (q_i^b \in E) \wedge (rq_i \notin E)\}$$

A transaction T_i has a *unanimous fate* in an execution E if

$$\mid fate(T_i, E) \mid \; = \; 1$$

at is, all polarities considered in the construction of $fate(T_i, E)$ are identical. This polarity is referred to fate of T_i, if indeed T_i has a unanimous fate.

An execution E is *semantically atomic* (SA) if for each transaction T_i:

- There is at least one subtransaction, $p_i \in T_i$ that has no recovery subtransaction in E; and
- T_i has a unanimous fate.

Observe that if a recovery subtransaction fails, the execution does not preserve semantic atomicity since t nsaction to which the subtransaction belongs cannot have a unanimous fate. We note without proof that mmit protocol (structured as prescribed in Section 5.2.1) that satisfies the unanimity condition ensures th ntually executions are SA. Compliance with the unanimity condition satisfies the first requirement in t finition of an SA execution. Our definition of semantic atomicity is an extension of the definition in [GM83] lude both compensation and retry.

The property of semantic atomicity is not prefix-closed. Consequently, as a non-SA execution progresses a re recovery subtransactions are executed it may become SA.

.2 Isolation of Recoveries for Composite Transactions

was intuitively explained in section 5.3, the notion of IR is concerned with the visibility of effects of transactio at do not have a unanimous fate. In standard transaction models, visibility is modeled by the *reads-from* confl ation (see [BHG87] for the exact definition). Since we do not deal with reads and writes, every conflict amo btransactions of different transactions represents the fact that the effects of the preceding subtransaction, visible to the subsequent subtransaction, p_i. Recall that causality and logical precedence are modeled dependency orderings within a transaction. Hence, it is appropriate to model the further propagation of t ects of t_j within T_i along these orderings. The notion of propagation and visibility of effects is made formal fining the *follows* relation.

Let $\xrightarrow{i}$ denote the transitive closure of the $<_i$ relation. A forward subtransaction p_i *follows* a subtransacti $(i \neq j)$ in an execution E if:

- $t_j <_E p_i$, and there is no s_k such that
  $t_j <_E s_k <_E p_i$. Or, if
- q_i follows t_j, and $q_i \xrightarrow{i} p_i$.

An execution *isolates recoveries* if there is no forward subtransaction of a particular transaction that follo btransactions of opposing polarities of another transaction. Formally:

Let s_j^b and $t_j^{\bar{b}}$ be two subtransactions of T_j that have opposing polarities in an execution E. Then an executi *isolates recoveries* (IR) iff whenever p_i follows s_j, then p_i does not follow t_j.

ne partial order models the logical precedences within a composite transaction. For brevity, a composite tran tion is often referred to just as a transaction.

The termination of composite transaction is coordinated using the commit protocol outlined in Section 5.: d employing either one of the decision rules described there. No synchronization points are assumed, howev nat is, a request for vote message from the coordinator, may be received after the local subtransaction has alrea rminated and released its resources. Consequently, there is no notion of global serializability for compos ansactions.

We subscript subtransaction names to denote the transactions they belong to, or the transaction they co spond to in case of recovery subtransactions. For example, p_i is a forward subtransaction [1] of transacti , rp_i is its corresponding recovery subtransaction, and s_i denotes either a forward subtransaction of T_i or rresponding recovery subtransaction. Regardless of whether s_i is a forward or a recovery subtransaction, y that s_i is a subtransaction of T_i.

We treat all conflicts among subtransactions as dependencies in the sense of Section 5.4.1 (i.e., we do n stinguish among read-write conflicts, write-write conflicts etc.). Observe, however, that there are no intr ansaction conflicts among subtransactions (except for the conflicts between subtransactions and their recove btransactions). This is because a transaction may have only one subtransaction at a particular site, and da ms at different sites are disjoint. Thus, to model intra-transaction dependencies that span across sites we ne additional notion. An ordering among two subtransactions of the same transaction is called a *dependen dering*. All the dependency orderings are inter-site. These dependency orderings model the logical precedenc w of information, causality and synchronization constraints among subtransactions of the same transacti at are imposed by its program. Regardless of the actual type of dependency modeled by $p_i <_i q_i$, we say th *invokes* q_i.

.1 Composite Executions

complete composite execution E over a set of composite transactions $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a partial order wi dering relation $<_E$ where

- $E = \cup_{i=1}^{n} T_i \ \cup \ rec$, where

$$rec \subseteq \cup_{i=1}^{n} \{rp_i \mid p_i \in T_i\}$$

  That is, E consists of the subtransactions iof the transactions in $\mathcal{T}$ and recovery subtransactions for a subs of these subtransactions.

- Each subtransaction in E has a polarity. Polarities are used below to encode the fate of a transaction in execution.

- $<_E \ \supseteq \ \cup_{i=1}^{n} <_i$.

- For any two conflicting subtransactions s_i, t_j, either $s_i <_E t_j$ or $t_j <_E s_i$.

- For any pair p_i, rp_i, of forward and recovery subtransactions, $p_i <_E rp_i$.

A *composite execution* is a prefix of a complete composite execution. Since all the executions, hereafter, a mposite, for brevity we refer to a composite execution merely as an execution.

[1] Even though we resort to a more conventional notation for subtransactions in Chapter 7 (T_{ij}, where the second index, j is a s me), here we prefer the p_i notation to avoid double subscripting.

Chapter 6

Atomicity of Composite Transactions

this chapter, we formally introduce composite transactions in the context of a DTM system. A formal definiti IR for composite transactions is given in Section 6.2. A corresponding protocol, referred to as the *polari otocol* is presented and proved correct in Section 6.3. Comments on the underlying model of this chapter a scussed in Section 6.4. Section 6.5 describes methods for incorporating actions that cannot be compensated-f retried into our paradigm.

A distributed database is a set of disjoint databases, where each database that is associated with a *site*. *btransaction* is an atomic transaction that consists of a totally ordered sequence of *accesses* to data items a gle site. Since our discussion, hereafter, is at the subtransaction level, the specifics of the accesses (i.e., wheth ey are reads, writes or other types of operations) are abstracted. To further justify this abstraction we assu e following regarding the interleaving of accesses to data items:

- **Serializability and Strictness.** The executions of accesses to data items are serializable and str [BHG87] at the subtransaction level. Strictness means that a subtransaction does not access a data ite before the previous subtransaction to access x terminates (commits or aborts).

The success or failure of a subtransaction (i.e., whether it committed or aborted) is encoded by a bina *larity*. For clarity, assume that if a subtransaction commits (aborts) its polarity is 1 (0). Observe that strictn needed so that a subtransaction p is assigned a polarity before subsequent subtransactions access the data ite cessed by p. The necessity of this requirement becomes clear later.

There are two kinds of subtransactions; *forward* subtransactions and *recovery* subtransactions. Each forwa btransaction is associated with a recovery subtransaction. We use p, q, o to denote forward subtransacti d rp, rq, ro to denote their respective recovery subtransactions. When we refer to either a recovery or forwa btransaction, we use s, t. The notation s^b (where b is either 0 or 1) denotes that s has polarity b. A polar posite to b is denoted $\bar{b}$. If a recovery subtransaction, rp, succeeds then rp's polarity is opposite to p's polari herwise, if rp fails, the polarities of p and rp are identical. A forward subtransaction and its successful recove btransaction always have opposing polarities. Intuitively, this represents the fact that a committed recove btransaction reverses the effect of its forward subtransaction.

Subtransactions accessing at least one data item in common are said to be *conflicting*. A recovery subtrans n accesses at least all data items accessed by its forward subtransaction. Therefore, a forward subtransacti d its recovery subtransaction conflict.

A *composite transaction* T_i, is a partial order with ordering relation $<_i$ where

- the elements of T_i are a fixed set of forward subtransactions; and
- there is at most one subtransaction of T_i at a particular site.

- The site executing the transaction crashes.
- The transaction was aborted intentionally, in order to resolve a deadlock, or for other reasons.
- A logical error in the transaction's code led to its abort (e.g., division by zero, attempt to violate integri constraint).

ne most simple form of retrying a transaction is re-executing its program. Following the first two types ilures, re-execution of the transaction may also fail. In case of a logical error that is state-dependent, the err ay occur again depending on the state of the database during the re-execution. Therefore, regardless of t use of the failure, we cannot require a retry by re-execution to succeed unconditionally as was required mpensation by the persistence of compensation requirement.

A more sophisticated retry transaction can examine the log records of the forward transaction and determi e cause for the failure. Based on this analysis, the retry transaction may take appropriate actions, there creasing its probability to succeed. Such a retry mechanism is similar to an exception handler whose actic e determined by the type of the failure. In addition, a retry transaction may invoke contingency actic LLR90, RELL90, BOH+91, C+89] if the failure analysis leads to the conclusion that mere re-execution is futi contingency action performs a task that is functionally equivalent to the task that was originally associat th the transaction.

If the semantics of a particular forward transaction are such that the unconditional success of this exa nsaction is crucial to the success of the entire transaction, then retry should not be considered as a recove tion for such a transaction (see Section 6.5). Similarly to compensation, retry is not universally applicab efer to Section 6.5 where incorporating of actions that cannot be retried into our framework is described.

Next, we illustrate the utility of retry in a DTM environment. As was alluded above, our basic paradigm is ablish a relaxed notion of atomicity given that subtransactions commit or abort in an uncoordinated mann ing semantics-based recovery. Relaxed atomicity, similarly to standard atomicity, offers two options for t al fate of a transaction. Establishing the option that parallels the standard Abort ('nothing') is obtained mpensating for all tentatively committed subtransactions. In a dual manner, we claim that the Commit opti ll') can be established by executing a retry subtransaction for all the failed subtransactions. The duality e two methods is illustrated by considering the case of a commit protocol employing a standard biased decisi le. If a local decision is reached prior to a global decision, the two decisions can differ only in case of a lo mmit. Thus, compensation can patch up relaxed atomicity. Conversely, if a global decision is reached prior ecting a local decision (i.e., prior to the local commit point) then the decisions can be incompatible only se of a global commit, and then retry establishes relaxed atomicity.

In the context of autonomy, i.e., multidatabases, retry has another importance. Compensation accommoda tonomy by allowing a site to commit in a tentative manner before a global commit decision is made. In al manner, retry facilitates forcing a global commit decision that is accepted before the sites themselves ha ysically committed the subtransactions. Such a capability to force a global commit decision is useful in ultidatabase environment [BST90].

One can argue that the localization of compensation principle holds regardless of the dependency classificati
the forward transactions. Once a forward transaction has executed, regardless of whether it had inter-depende
s or not, the traces it leaves in the form of local logs at the different sites look the same for the purposes
mpensation. It is anticipated, however, that it is going to be easier to enforce the localization principle f
obal transactions that have no dependencies. When a global transaction is semantically decomposed, each
subtransactions has clear semantics. Thus coming up with independent compensating subtransactions shou
se less difficulty. It is going to be harder to enforce the localization principle when syntactic decomposition
ed. Some global information might be needed in order to assign the individual compensating subtransactio
e relevant semantics. We postpone to future research a more precise analysis of the interplay among the typ
global transactions and the validity of the localization principle.

.6 Two Specific Solutions

iapters 6 and 7 provide two specific methods for obtaining relaxed atomicity and ensuring IR. To give the reac
oroader perspective on this part of the dissertation, we provide in this section a brief preview and comparison
e two methods. Chapter 6 talks about composite transactions and is based on [LKS91b]. Chapter 7 describ
Optimistic 2PC (O2PC) protocol and is a variation of [LKS91a]. For clarity, in both chapters all transactic
e assumed to be sensitive; that is, IR needs to be enforced for all of them.

The work in Chapter 6 is characterized as follows:

- The transaction model is an advanced model [tm-91, BOH$^+$91].
- Both decisions rules are considered, and hence both retry and compensation are employed.
- There are dependencies among subtransactions and they are modeled by a partial order. Consequently,
is enforced in an incremental manner.
- IR is enforced on data items granules.
- Subtransactions may have no synchronization points, and hence global serializability is not an issue.

The work in Chapter 7 is characterized as follows:

- A conservative and standard transaction model is used.
- Only compensation, and not retry, is relied-upon, since only the standard biased decision rule is considere
- There are no dependencies among subtransactions. Therefore, IR is imposed using a validation method.
- IR is enforced at a site granularity.
- Subtransactions have synchronization points.

is possible to combine features from either method and come up with a synthesized protocol.

.7 Retry Transactions

nilarly to a compensating transaction, a retry (sub)transaction is coupled with a forward (sub)transactic
rst, we discuss briefly how a retry transaction can be constructed. Retry is initiated based on the premise th
e forward transaction has failed. There are few possible reasons for a transaction failure:

.5 Localization of Compensation

ur basic idea is to promote the notion of tentative local commitment of subtransactions that can be undo compensation if the global transaction fails. For that end, we have coupled compensation activities with su nsactions. A question that should be addressed is how to treat the collection of compensating subtransactio at is associated with the global forward transaction. Answering this question has ramifications on defining t rrectness of executions with compensation in a distributed environment.

For answering this question, we point out the following special features of compensation in a distribut stem:

- Compensation, as a recovery activity, is an *after the fact* activity. That is, the forward transaction h executed, and compensation is carried out based on its effects. Similarly to standard undo, the compensati subtransaction is guided by the *local* log in determining which operations with which arguments should applied. Therefore, there is no single global program that drives the individual compensating subtransactio at the different sites.
- By the persistence of compensation requirement (Section 4.1.4), each site automatically guarantees t eventual successful termination of the local compensating subtransaction. Consequently, there is no ne to use a commit protocol to coordinate the termination of the compensating subtransactions. Each lo compensating subtransaction can terminate independently.

In light of the above, we advocate the principle that compensating for a global transaction need not ordinated as a global activity. That is, there is no such entity as a global compensating transaction. Instea mpensating subtransactions of a single global transaction should be treated as a collection of almost independe cal transactions. More precisely, the *localization of compensation* principle asserts that:

- The execution of a compensating subtransaction does not depend on the execution of its siblings.
- The termination of a compensating subtransaction need not be coordinated with the termination of siblings.

This principle can be better understood by making the analogy to traditional DTM systems. There, wher obal transaction is aborted, each site is responsible for undoing the local subtransaction. Only the initiati the recovery at each site is coordinated, namely it is prompted by the receiving of an abort decision messa om the 2PC coordinator of the global transaction. We extend this principle of localized recovery for recove compensation. A similar idea, referred to as recovery non-interference, is reported in [LYI87]. In suppo localization of compensation, we also cite [Vei89, map89] where a large-scale, commercial application that edicated on a similar principle, is described.

Being a recovery activity, compensation is inherently considered as an overhead function. Therefore, the co compensation should be kept at minimum. By localizing compensation, the expensive communication for t ordination of a global activity is avoided. Also, observe that avoiding the need to coordinate the termination e compensating subtransactions is crucial. The alternative of using an atomic commit protocol for the glob mpensating activity would have contradicted the initial objective of alleviating the problem associated wi omicity in a distributed environment.

The validity of the localization principle and the difficulty in enforcing it depends on the decomposition a pendency classification of the forward transaction. In what follows, we provide a qualitative analysis of t erplay among these notions.

s no orderings, IR is enforced by *validation* method. The coordinator validates the execution of all subtransa ns once they return results and are ready to terminate. In the validation, the coordinator ascertains that t btransactions have not observed both compensated-for and committed effects of other transactions (the speci gorithms are presented in Chapter 7). On the other hand, when there are orderings, the coordinator has onitor the execution of the subtransactions as it progresses *incrementally* to make sure IR is preserved.

We note that is not surprising that the issue of the shape of the program of transactions surfaces again, aft ing discussed in Chapter 2 in the context of single-transactions compensation.

4.2 Decomposition into Subtransactions

the decomposition of a global transaction to local subtransactions, we refer to the aspect of the granular the operations that are shipped to a site for execution. Another way to look at the decomposition dimensi in terms of the interface a site exports for global transactions.

The two extreme decomposition methods we discuss are *semantic* and *syntactic*. Similar classification c found in [Joh90]. Using a syntactic decomposition, the set of primitive requests of a global transaction to rticular site constitute the local subtransaction at that site. Each subtransaction can be viewed as an arbitra llection of reads and writes against the local data. That is, no predefined semantics is associated with btransaction. This model is elaborated in [CP87] and is the standard model in distributed databases. Syntac composition is also considered as the general framework in the multidatabases context [BS88, BST90].

On the other hand, using semantic decomposition, each global transaction is decomposed into a collecti local subtransactions, each of which performs a semantically coherent task. The subtransactions are select om a well-defined repertoire of procedures forming an interface at each site in the distributed system. btransaction may include more than one procedure, but still the significant aspect of associating well-defin d pre-determined semantics with each subtransaction is invariant. This decomposition is suitable for a federat stributed database environment [fdb87].

The distinction between the two models is obvious once semantic compensation is introduced. Our wo plies to both models; however, fitting the ideas in each framework is bound to be different, and probably eas hen semantic decomposition is employed.

4.3 Synchronization Points

btaining global serializability in a locking-based DTM system is typically based on synchronized release of loc order to enforce a two-phased locking (2PL) [BHG87] discipline over all accesses of a global transaction, ould be guaranteed that a lock is not released at one site prior to acquiring a lock at another site by the sa nsaction. This can be achieved through the synchronization that is introduced by the 2PC protocol. It sumed that the coordinator of T_j initiates the 2PC protocol only after it has received acknowledgements for T_j's operations. Therefore, when the coordinator initiates the 2PC protocol by sending the messages requesti tes (known as VOTE-REQ, or PREPARE) messages), T_j has surely obtained all the locks it will ever need. cks are released only after the VOTE-REQ message has been received, 2PL and thus serializability is obtain is combined 2PC/2PL protocol is very common (e.g., R* and CAMELOT [MLO86, Duc90]) and is referr in [BHG87] as *distributed 2PL*. We refer to this coupling of the beginning of the release phase of 2PL and t ceipt of the VOTE-REQ message as *synchronization point*. When a subtransaction releases its locks prior to ceiving the VOTE-REQ message, we say that it does not have a synchronization point.

Executions that are IR are formally defined in Chapters 6 and 7. The definitions in these chapters differ d the different underlying transaction models. Protocols that guarantees IR are also devised in these chapter

.4 Taxonomy of DTM Models

this section we review certain aspects of DTM that are relevant for the development of our ideas. We classi TM models along the dimensions of *decomposition* of a global transaction to local subtransactions, and *depe ncies and orderings* among the subtransactions. We also comment on methods for combining atomicity a nchronization concerns in DTM systems. The categories we define in this section are referred to later in t ssertation.

4.1 Dependencies and Orderings among Subtransactions

ne program of a transaction induces certain *dependencies* among the operations that implement the progra ne dependencies can result from either control or data flow of the transaction program. For instance, a con onal statement requires evaluating the condition before executing the branches. That is, the primitive operatio at implement the branch depend on the operations that are used for the condition evaluation. Likewise, signment statement induces a dependency between the reading of the right-hand-side and the update of the le nd-side. In more advanced transaction models, where the emphasis is on flexibility and expressibility, depende s can result from a variety of other reasons, e.g., causality, and synchronization constraints [tm-91, BOH$^+$9 global transaction, like an ordinary transaction, is driven by a program that induces certain dependenci pically, a *coordinator* is assigned the task of executing this program by spawning remote requests. Usually, tl ordinator also plays the role of the commit-protocol-coordinator for the particular global transaction [CP8 hen mapped for execution on the underlying distributed system, the global transaction is decomposed to l subtransactions. We choose to disregard all intra-subtransaction dependencies and narrow our attention e more interesting inter-subtransaction dependencies. Such dependencies are modeled in what follows by rtial order over the subtransactions. These issues of dependencies among subtransactions of the same glob nsaction are discussed at length in [DE89, ED89].

One simple case of global transaction structure is when the program is devoid of an elaborate control flow a s no dependencies among its constituent subtransactions. For example, consider a transaction that executes nple SQL-like [KS90] statement in the form of:

select $*$
from R

nere the relation R is fragmented in several sites. Mapping such a global transaction to the distributed e ronment results in an unordered set of subtransactions. The coordinator of such a transaction spawns t btransactions at the relevant sites in no particular order, and waits to gather all the results. This case ferred to in [ED89] as subtransactions with no *value dependencies*. On the other hand, mapping a dependen nong two subtransactions onto the underlying distributed architecture may require the coordinator to spav e dependent transaction only after the dependent-upon subtransaction has returned its result (since the latt pplies an input parameter for the former, for example).

The reader should be aware to the different execution pattern in these two cases. When there are no depe ncies, the coordinator spawns all the subtransactions and waits for them to return, whereas when there a pendencies the execution progresses incrementally at the different sites as dictated by the coordinator. T esence of orderings (that are the consequence of dependencies) among subtransactions, and lack thereof, disti ishes the methods presented in Chapters 6 and 7 in terms of enforcing IR. Namely, when a global transacti ne sites and aborted subtransactions at other sites. Problems may arise when the effects of such non-aton insactions become visible, thereby affecting other transactions. Specifically, there are transactions that shou t be affected by both failed (or compensated-for) and successful subtransactions of the same transaction. ist isolate such failed, non-atomic transactions until all the recovery subtransactions are executed and seman micity is obtained. We refer to this requirement as *isolation of recoveries*[2] (IR). We say that an executi IR to informally mean that no transaction in that execution observes both compensated-for effects as w committed effects of other transactions. In this section, we motivate this requirement and postulate it as rrectness criterion in the context of relaxed atomicity.

First, we argue that IR is indeed beneficial in excluding unacceptable executions by illustrating an examp nsider a global transaction T_1 which transfers funds from site 1 to site 2. The decomposition of T_1 into lo btransactions is simply:

- T_{11} – debit by amount a
- T_{12} – credit by amount a

sume that in a particular execution under our generic commit protocol (Section 5.2.1) it happens that T orts, whereas T_{11} commits locally. Compensation for T_{11} includes crediting by the amount a. Consider insaction T_2 that performs an audit at the two sites, by reading the balances at each site. The execution at t o sites is depicted below (each line represents the serialization order at a site from left to right).

$$\begin{array}{lllll} S_0: & T_{11} \quad T_{21} \quad CT_{11} & & & :S_1 \\ S_2: & T_{12} & & T_{22} & :S_3 \end{array}$$

e states S_0, S_2 denote the initial states of the accounts at sites 1 and 2, respectively. Likewise, the states S_1, note the corresponding final states. Clearly, in this scenario, T_2 reads a globally inconsistent state, where t iount a is unaccounted for.

Recall that based on the notion of $\mathcal{R}$-commutativity, being affected by a successful subtransaction that is lat mpensated-for is permitted (e.g., T_{21} being serialized after T_{11} in the above example). Thus, one might arg at with the aid of a proper relation $\mathcal{R}$, compensation may be designed to rectify the above anomaly. Ne refute this argument. Let $(T_{11} \circ CT_{11} \circ T_{21})(S_0) = S_4$. Then, although CT_{11} and T_{21} $\mathcal{R}$-commute, and as sult $S_4 \; \mathcal{R} \; S_1$, this does not change the fact that S_1 and S_3 do not satisfy the global consistency constraint intaining consistent total balances. Thus, the anomalous situation where T_2 is affected by both compensate and locally committed subtransactions cannot be rectified by $\mathcal{R}$-commutativity. The problem arises becau e relation $\mathcal{R}$ is based on local predicates alone, and it guarantees nothing regarding global relationships amo ta items at the different sites.

Transactions such as T_2 in the above example, are characterized by requiring a *global* consistency constrai hold on the data they access at multiple sites. Such transactions are referred to as *sensitive* transactions. T gree of atomicity guaranteed by $\mathcal{R}$-commutativity is not sufficient for sensitive transactions. We defer as futu search defining a clearer characterization of sensitive transactions. In chapter 6 and 7, for clarity of expositic assume that all transactions are sensitive.

An analogy between isolation of recoveries and serializability is in place. Serializability makes concurre ecutions transparent to the transactions. Likewise, isolation of recoveries makes non-atomicity transparent insactions. The importance of our study of IR as a correctness criterion is underlined by the growing populari advanced transaction models that are based on semantic atomicity [GM83, GMS87, AGMS87, KR88, Reu i89, GMGK$^+$90, tm-91, BOH$^+$91], and by the lack of specific correctness criteria in this domain.

[2] The choice of the name is intentional since IR as presented in this chapter is an extension for global transactions, of Constrain Section 2.3.

The work in this direction of relaxed atomicity, as opposed to the traditional atomicity, has not matured y
ie precise guarantees of such transaction models have not been examined to date. In particular, the attracti
ea of using relaxed atomicity in a distributed setting has not been carefully examined so far. Chapters 6 anc
e dedicated to formal treatment of relaxed atomicity and the ensuing correctness issues in a DTM setting.

2.1 A Generic Relaxed Atomicity Commit Protocol

e adopt and extend the notion of semantic atomicity mentioned above for DTM systems. First, we adopt t
nvention that a multi-site transaction, referred to also as *global* transaction hereafter, is decomposed into sing
e subtransactions. The commit protocol for global transactions proceeds as follows. A site decides whether
cal forward subtransaction commits or aborts *without coordination* with other sites executing subtransactic
behalf of the same transaction. Once this decision is made, all the local resources the subtransaction hol
e released at once. We say that the site makes a *local (commit) decision*. A centralized *coordinator* initia
commit protocol by requesting these decisions from the sites that executed subtransactions on behalf of t
-be-committed transaction. The decisions are cast as *votes* in the first phase of the commit protocol. T
ordinator gathers the votes and decides whether to commit or abort the transaction according to a decisi
le whose nature is explained shortly. This *global decision* is conveyed to the different sites in a second pha
the commit protocol. In case of a discrepancy between a local decision and the global decision, a recove
btransaction is executed at the local site. Namely, if the local decision was commit and the global one is abo
en the local subtransaction is *compensated-for*. Conversely, a local subtransaction is *retried* if the global decisi
commit and the local decision was abort. (We expand on retry in Section 5.7). Notice that semantics-bas
covery is done on a subtransaction basis. That is, each forward subtransaction is associated with a retry
mpensating counter-part (more on this issue in Section 5.5). The recovery subtransactions ensure convergen
a unanimous outcome at all sites despite the uncoordinated local decisions.

Any rule governing the decision making by the coordinator must conform to the *unanimity* requirement:

- **Unanimity.** If all votes are identical then the decision must be unanimous with the votes.

decision rule can be either *biased* or *arbitrary*. In standard atomic commit protocols, the following bias
cision rule is used:

- **Biased Decision.** If at least one of the sites votes to abort, then the decision is abort.

arbitrary rule can be based on quorum, majority or other principles that conform with the unanimity requi
ent [1]. For instance, a transaction may be considered successful if a certain subset of its subtransactions succee
ie specifics of the arbitrary decision rule are abstracted from our discussion. The main distinction between t
o rules is the possibility of reversing a local abort decision by a retry subtransaction. This option is lacking
e biased case, and is present in the arbitrary case.

The description given above serves as a common framework for both Chapters 6 and 7.

.3 Isolation of Recoveries

ie concept of atomicity is intended to mask failures by creating a virtual failure-free system in a failure-pro
vironment. When relaxing atomicity, as we propose to do, we must make sure that failures do not becor
sible. Since semantic atomicity is an eventual property (i.e., eventually all locally committed subtransactic
ll be compensated for), there are time intervals where transactions have locally committed subtransactions

[1] We do not deal with optimizations to the commit protocol that are possible when orderings are imposed among the subtransacti
g., like the linear 2PC protocol [Gra78]).

enomenon where transactions may be delayed for *unbounded* periods. In the context of 2PC, blocking impl at if the coordinator for a transaction, or a communication link to that coordinator, fails in a certain critic ne, some other transactions at active sites are delayed until the failure is repaired.

Another severe difficulty arises when atomic commitment is considered in the context of multidatabase syste here several sites are integrated to create a cooperative environment (see [hdb90] and the references there). T al of the integration is to support global transactions by dividing them into local subtransactions that a ecuted at the different sites. In a multidatabase, the individual sites comprising the integrated system may u terogeneous database management systems. The sites may belong to distinct, and possibly competing busine ganizations (e.g., competing computerized reservation agencies). In such systems the local *autonomy* of t dividual sites is crucial. It is undesirable, for example, to use a protocol where a site belonging to a competi ganization can intentionally or innocently block the local resources. One of the flavors of local autonomy fined as the capability of a site to abort any local (sub)transaction at any time before the (sub)transacti rminates. Employing the 2PC protocol, a site enters a *prepared* state if it votes to commit a transaction ra78]. Once in this state, a site becomes a subordinate of the external coordinator, and it can no long ilaterally determine the fate of the local subtransaction of T. Therefore, local autonomy is sacrificed once C protocol is imposed on the integrated sites.

In summary, the fundamental problems associated with an atomic commit protocol in a DTM system are:

- Lengthy delays are introduced by the need to coordinate the termination of the distributeed subtransactio
- A potential for the undesirable phenomenon of blocking is introduced.
- The autonomy of individual sites is compromised once the protocol is imposed on the distributed syste

ese problems are exemplified by the popular 2PC protocol. In our work on semantics-based recovery in DT stems we attempt to solve, or at least alleviate these problems. Our work is concerned with a generic DT odel. Thus, we do not deal directly with the intrinsic problems of multidatabases. However, the results can stantiated, and their relevance is amplified, where this particular case of a DTM system is considered.

.2 Outlining a Solution: Relaxing Atomicity

cing the impossibility results regarding atomic commitment in distributed systems [Ske82, BHG87] it is evide at in order to overcome the above problems, a trade-off must be exploited. Our thesis is to *relax* the guarantees nsaction atomicity, thereby obtaining a handle for solving the difficult problems we are faced with. Promine evious proposals in the context of relaxed atomicity are sagas [GM83, GMS87], and their generalization ultitransactions [GMGK$^+$90, BOH$^+$91]. These proposals do not consider the atomicity problem in distribut tting, but rather the similar problem of atomicity of long-duration transactions. Essentially, the idea in the oposals is to decompose the coarse unit of a transaction into finer subtransaction units. Subtransactions comm abort independently, without coordination with other subtransactions of the same transaction. Resources he the subtransactions are released as soon as the subtransaction terminates, without waiting for the terminati the entire transaction. Therefore, atomicity of the whole transaction is given up for a weaker notion referred *semantic atomicity* [GM83]. Each subtransaction, T_{ij}, is associated with a compensating subtransaction, wh sk is to undo semantically the effects of T_{ij} in case the entire transaction aborts. Instead of the standard a nothing atomicity, semantic atomicity guarantees that either all subtransactions commit—and then the ent nsaction commits, or that all subtransactions that committed in a tentative manner are compensated-for, e entire transaction is to abort.

Chapter 5

Distributed Transaction Management: Preliminaries

ving studied compensation for an individual transaction, we turn our attention to semantics-based recove
composite transactions—transactions that are composed of several simple subtransactions. Since we draw o
tivation from the use of composite transactions in distributed systems, distributed transaction manageme
TM hereafter) is the focus of this chapter and the remainder of the dissertation. We concentrate on compos
nsactions, whose constituent subtransactions execute at different sites of a distributed system.

Composite transactions are formally introduced in the next chapter. In this chapter, we first outline the ba
blem of atomicity in DTM systems in Section 5.1. We sketch our solution in a concise and informal manner
ction 5.2. Our approach exploits a trade-off as it solves the problem by relaxing standard atomicity. Relax
omicity gives rise to a host of issues that are referred to as *isolation of recoveries* (IR). This pivotal conce
informally motivated and presented in Section 5.3. In Section 5.4 we classify DTM models along dimensio
at are relevant for our results. In Section 5.5 we introduce the localization of compensation principle, whi
ves as a basis for the use of compensation in the context of DTM. Chapters 6 and 7 deal with specific metho
at are based on the general paradigm described in this chapter. Section 5.6 briefly contrasts this two metho
ereby giving the reader an overview of Chapters 6 and 7. Finally, the other semantics-based recovery meth
retry, which is made use of in Chapter 6, is introduced in Section 5.7.

.1 The Problem

hen transactions access data items at multiple sites of a distributed system, atomicity is accomplished
ploying an *atomic commit protocol*. The two-phase commit (2PC) protocol [Gra78] is the most comm
omic commit protocol, and it is widely used in distributed transaction management systems. In this protocol
lti-site transaction is associated with a coordinator that gathers votes from the participating sites as to wheth
commit or abort the transaction. Based on these votes, the coordinator makes a decision and transmits it
participating sites. The receipt of this decision message must precede the release of all resources held by t
nsaction that is involved in the 2PC protocol. Consequently, all other transactions contending for the resourc
ld by the transaction in question must wait until the decision message has been received. If the execution tim
the actions of a multi-site transaction differ from site to site, these delays make the execution duration of
tions at all sites equally long.

It is well known that there is no atomic commit protocol that is not *blocking*, in a distributed system that
bject to (fail-stop) site failures, and failures in communication links [Ske82, BHG87]. Blocking is the undesira

ould be established [GM83]. The compatibility will represent the various restrictions on exposing data amo
pes of transactions. More work needs to be done on the subject of building such a type-specific lock manag

The various restrictions on the dependent transactions (e.g, having fixed or linear programs, $\mathcal{R}$-commutativit
dicate that compensation is a history-based activity. That is, the applicability of compensation depends
e execution, and in particular on the nature of the dependent transactions. In essence, it is impossible
arantee proper compensation (i.e., in a manner that ensures a degree of atomicity and consistency) regardl
the execution, and independently of the dependent transactions. Instead, there is a spectrum of less extre
ssibilities that constitute a trade-off. The fewer restrictions are imposed on transactions, the more involved t
mpensation is bound to be, and the weaker the resultant atomicity guarantee will be. It is probably possi
custom-design compensating transactions that are very specific to an application, and require an intrin
owledge of its semantics. Ideally, however, there should be a generic mechanism that activates the necessa
mpensatory actions as required, similarly to the way a traditional undo mechanism operates. If the forwa
nsactions are structured carefully, then compensation can be simple and largely applicable. For instan
agine a library of pre-defined forward routines, each of which associated with a counter-routine. All forwa
nsactions are composed out of this well-defined repertoire of routines. Thus, it is possible to guarantee, by eith
mal verification or exhaustive testing of all possible combinations, that the counter-routine always compensa
forward counterpart properly regardless of the dependent transactions. Under these circumstances, it is possi
automatically extract the actions of a particular compensating transaction from the log.

.2 On the Design of Compensating Transactions

nce compensation is an application-dependent activity, it is hard to identify universal principles for the design
mpensating transactions. Nevertheless, we have provided a formal basis for the design and use of compensatio
is formal basis emphasizes that while semantically undoing the forward transaction, certain predicates expre
g general consistency constraints and specific properties established by the dependent transactions should
eserved. The examples in Chapter 3 and the comprehensive example in [SLKS91] shed some light on this form
sis by applying it to actual scenarios. In this section we generalize the examples and provide insights regardi
e design of compensating transactions that cannot be captured in formal terms.

Compensating transactions are intended to counter the phenomenon of cascading aborts by undoing a tra
tion without undoing its dependents. This purpose, and the cascading effect itself, can be traced in the desi
a compensating transaction. A reasonable design rule is to first perform an actual logical undo, and th
store consistency, or establish other desirable properties. This restoration activity is the result of undoing t
mpensated-for transaction while leaving the dependent transactions intact. For instance, consider the compe
tion illustrated in Section 3.3, where first the tuple is deleted, and then the average computation is amended. C
nsider the example in Section 7.1 [SLKS91], where first the erroneous track is removed and then the positioni
the gun is corrected. The pattern is repeated: by undoing a transaction with dependents, some inconsistenc
se, and some desirable properties are violated. Thus, after undoing, a restoration phase is called for.

The task of re-establishing the desirable predicates by the compensating transaction depends on the existen
d the nature of the dependent transactions. Typically, if there are no dependent transactions, logical undoi
all that compensation performs (e.g., referring again to the example in Section 3.3, just removing the tuple
fficient if the average computation does not take place). Therefore, the code of a compensating transaction mu
eck the log and determine whether the forward transaction has dependent transactions and act accordingly.

A form of cascading undoing is sometimes unavoidable. Often, there are semantic dependencies among tra
tions, such as causality, that necessitate compensating in a cascading manner for transactions dependent on t
ginally compensated-for transaction. For instance, referring back to the airline reservation example in Secti
., the special meal order of the canceled flight reservation should be canceled, too. There are several alternati
handle causally-dependent activities. First, such dependent (sub)transactions should be encapsulated wit
single sphere of recovery. Namely, this is a case for a composite transaction in the form of a nested transa
n [Mos87], a saga [GMS87], or its generalization — a multi-transaction [GMGK$^+$90, KR88, Reu89]. In su
odels of composite transactions, the effect of a cascading abort (or cascading compensation for that matter)
ntrollable, predictable, and confined to the boundaries of a single transactional unit. A competing approa
the active, or triggered, model for transaction dependencies [DHL90, MD89, C$^+$89]. Causality is modeled
ggering the dependent transaction when the causing event occurs. A good example here is a cancellation
servation in an airline reservation system which is handled as a compensating transaction that triggers t
nsfer of pending reservation from a waiting list to the confirmed list. These issues, however, fall outside t
ope of this dissertation.

Theorems 1, 2 and 3 indicate that the externalization (i.e., exposing updated data items by releasing lo
example) of uncommitted data items should be done in a controlled manner if a degree of atomicity is
portance. That is, uncommitted data should be externalized only to transactions that satisfy the requireme
ecified in the premises of the theorems. In the context of locks, locks should be released only to qualif
nsactions, that is, those transactions that do satisfy the requirements. Other transactions must be delay
d are subject to the standard concurrency control and recovery policies. This restriction concerns also actio
at are not-compensatable (i.e., the real actions of [Gra81] that were mentioned in Section 2.3). In gener
classification of transaction types is necessary. Further, a notion of compatibility of these transaction typ

eas from [HMS88], which describes an extensible logging service, can be incorporated for the design of the loggi chitecture. It is important to note that the technology trend of large main memories can support fast rando cess to the log, by storing at least the tail of the log in main memory [Bit86, DKO$^+$84].

Yet another problem is concerned with supporting compensation for transactions whose log records span gthy log interval. Such long interval can cause difficulties in terms of reusing log space. A log compacti echanism must exist under such circumstances.

Typically, the portion of the log in between the forward and the compensating transactions is the relevant se ent of the execution. However, sometimes, compensation has to look at even earlier execution. If compensati to amend a value that tracks an entity based on periodic recordings (e.g., tracking a mobile target), extrap ion based on past values can be used as the basis for the compensation [SLKS91]. Under these circumstanc rlier execution, prior to the forward transaction must be considered by the compensation.

1.3 Explicit Invocation of Compensation

far it has been assumed that a compensating transaction can be invoked internally by the recovery manager onsequence of the abort of an externalized transaction. In a system that supports compensation, it is possil allow users to invoke a compensating transaction explicitly in order to cancel the effects of a committ nsaction in the same manner as regular transactions are invoked. Such a feature can be useful in the followi nario. Suppose that a transaction was committed "erroneously." By committed erroneously, we mean th m the system's point of view there was nothing wrong with the committed transaction. However, extern asons, that were discovered later, rendered the decision to commit the transaction erroneous. Being aware ese circumstances, the user may invoke the proper compensating transaction that will automatically amend t uation.

1.4 Persistence of Compensation

was noted earlier, we should disallow a compensating transaction to be aborted either externally (by user, or plication), or internally (e.g., as a deadlock resolution victim). A simplistic (and usually unsatisfactory) soluti the problem of deadlock resolution is the notion of *golden transactions* in system R [GM$^+$81]. By running or e golden transaction at a time, the system can always avoid choosing these privileged transactions as deadlo ctims. A more suitable solution is to support automatic restarting of compensating transactions if they fa ch a mechanism can be used for making compensating transactions persistent across system crashes. Followi crash, all interrupted compensating transactions should be treated as *pending actions* that must be redo e mechanism for implementing this strong persistence can be based on resuming a compensating transacti om a savepoint [MHL$^+$90], or on restarting. In case of resuming from a savepoint, the internal state as w the concurrency control information of the compensating transaction must be saved in log records. In case starting, the interrupted compensating transaction has to be undone first, and then automatically restarted. phasize that the principle for recovery of compensating transaction is that once a begin-transaction record T appears in stable storage, CT must be completed eventually. An implementation along the lines of the ARI stem [MHL$^+$90] can support the persistence of compensating transactions across system crashes. In ARI do activity is logged using Compensating Log Records (CLRs). Each CLR points (directly or indirectly) next regular log record to be undone. It is guaranteed that actions are not undone more than once, and th do actions are not undone even if the undo of a transaction is interrupted by a system crash.

Chapter 4

Practical and Design Issues

this chapter we discuss several issues that should be addressed in order for compensation to be of practic e. Among other issues, we discuss specific design rules for compensating transactions.

.1 Practical Issues

ompensation is a powerful method. However, executing it can be expensive unless adequate special support is provided.

1.1 Logging Scheme

equiring a compensating transaction to succeed unconditionally (the persistence of compensation requiremen plies that design of a compensating transaction is a complex and application-dependent task. The fact that t mpensating transaction always executes *after* its forward counter-part must be used to alleviate this difficul sentially, the forward transaction should record enough semantic information, e.g., in the form of log recor order to guide the proper execution of the compensating transaction. Therefore, it is likely that some for operation logging will be used [HR83]. Operation logging is the logging scheme used in conjunction wi e semantically-rich logical undo methods. Another reason why logging operations, rather than logging physic anges, is preferable is that it enables detecting dependencies among transactions by analyzing the log sequenc he necessity of logging enough information to enable detecting dependencies among transactions (e.g., loggi ad accesses) is discussed in [PKH88].

1.2 Log Retrieval

nce compensating transactions are envisioned to be driven by a scan of the log, it is important to provi ficient on-line access to the log information. Without a suitable logging architecture, the accesses to the l ght translate to I/O traffic that would interrupt the sequential log I/O that is performed on behalf of executi ward transactions. A related problem is the efficiency of the log scan which impacts the performance of t mpensating transactions themselves. Since compensating transactions rely on qualified retrieval of log recor random access to the sequentially written log device must be supported. In order to facilitate such retrie ficiently, some (indexing) structure must be imposed on the sequence of log records. Stable memory [CKKS n be used for creating and maintaining this structure. The stream of log records can be post-processed in t ilure-proof random-access memory before this stream is oriented to secondary storage. We briefly mention so rk that has been done in providing efficient access to sequential streams of data (i.e., logs) [LC87, FC87, DST

an illustration: The compensated-for transaction extends a file, or is allocated storage, and the additio
ace is used by other transactions; the compensated-for transaction frees space that is later allocated for ot
nsactions; the compensated-for transaction inserts a record to a B-tree that causes a split of a node, and ot
nsactions use the new nodes; the compensated-for transaction updates the free space information mechani
the storage manager (percentage of occupied space in a page, etc.) and other transactions update the sa
ormation. (See discussion on these issues in [ML89, Moh89]). We note that, in all the above storage managem
amples, although effects are exposed to transactions, they are not exposed to users.

.3 The Average Computation Example

this example we illustrate the use of semantics in the design of a compensating transaction. Consider t
lowing transactions:

- T: A long-duration transaction, one of whose operations inserts a tuple t into a relation R, with a nume
attribute A.
- T': A transaction that scans the relation R, counts the tuples and computes the average of their value
the attribute A. For reference purposes, assume that T' stores the count in N, and the result of the avera
computation in *average*.

early, $T' \in dep(T)$. Since T is a long-lived transaction, the insertion operation is exposed early, before
mmits. In particular, the inserted tuple t was counted by T' and considered in the average computati
sume that T has to abort. According to the conventional recovery approach, T' would have to be abort
a cascading manner. It is undersirable to abort T' since it is an expensive transaction that scans the ent
relation. An alternative scenario in which aborting T' would have been undesirable is one where T' must
ecuted fast to provide the average result as an assessment for a decision support application. For these reaso
is highly undersirable to abort T' once T aborts. Fortunately, a simple compensation can help:

- CT: We describe the part of CT that handles the tuple t and the average computation. The notation t[
denotes the value of tuple t on attribute A.

```
. . .
delete(t)
oldN := N
N := N - 1
average := (oldN*average - t[A])/(oldN - 1)
. . .
```

us, instead of aborting T' and redoing its task, we were able to amend the situation easily, based on t
mantics of computing averages. Compensation in this case yields an atomic execution since all the effects of
canceled. Observe how CT first logically undoes the insertion of t and the increment of N, and then resto
value of *average* appropriately. It is assumed that CT checked the log of the execution and discovered th
depends on T and acted appropriately. Another lesson from this example is that exposing uncommitted da
ust be done in a controlled manner. Namely, only these transactions for which enough semantic informati
ists can read the exposed uncommitted data; e.g., only to the average computing class of transactions in th
se.

```
(rs + x) <= seats then rs:=rs+x
                   else reject:= reject+1
```

The consistency constraint Q in this case is:

$$Q(S) \;\; iff \;\; S(rs) \leq S(seats)$$

sume:

$$S = \{seats = 100, rs = 95, rejects = 10\},$$

$$T = \mathbf{reserve(5)}\;,\; dep(T) = \{\mathbf{reserve(3)}\}$$

t the execution be $X \equiv X_T \circ X_{dep(T)} \circ X_{CT}$ where CT is defined by Definition 12. We would like to ha er X: $S' = \{rs = 95, rejects = 11\}$, that is, T's reservations were made and later canceled by running C d $dep(T)$'s reservations were rejected. And that is exactly what we get by our definition. Observe how T ervations were canceled, but still its indirect impact on $rejects$ persists since T caused $dep(T)$'s reservatio be rejected.

Hence, this example demonstrates an execution that is not atomic but is nevertheless intuitively acceptab d the transaction in $dep(T)$ been executed alone, it would result in successful reservations. In formal tern can define a relation $S' \; \mathcal{RS}''$ to hold only if Q is satisfied by both states, and then state that the executi $\mathcal{R}$-atomic. Notice how in this example the operation of CT can be implemented as inverse of T's operati ddition and subtraction). The less interesting case, where there are enough seats to accommodate both T a p(T), also fits nicely. In this case CT's subtraction on the entity $seats$ commutes with $dep(T)$'s addition s entity.

.2 Storage Management Examples

ne following example is from [MGG86], though the notion of compensation is not used there. Consider tra tions T_1 and T_2, each of which adds a new tuple to a relation in a relational database. Assume the tup ded have different keys. A tuple addition is processed by first allocating and filling in a slot in the relatio ple file, and then adding the key and slot number to a separate index. Assume that T_i's slot updating (S_i) a dex insertion (I_i) steps can each be implemented by a single page read followed by a single page write (writt tp], $w_i[tp]$ for a tuple file page p, and $r_i[ip]$, $w_i[ip]$ for an index file page p).

Consider the following execution of T_1 and T_2 regarding the tuple pages tq, tr and the index page ip:

$$< \; r_1[tq],\; w_1[tq],\; r_2[tr],\; w_2[tr], r_2[ip],\; w_2[ip],\; r_1[ip],\; w_1[ip] \; >$$

This is a serial execution of $< S_1 \;,\; S_2 \;,\; I_2 \;,\; I_1 >$, which is equivalent to the serial execution of executing d then T_2. Assume, now, that we want to abort T_2. The index insertion I_1 has seen and used page p, which w itten by T_2 in its index insertion step. The only way to abort T_2, without aborting T_1 is to compensate for rtunately, we have a very natural compensation, CT_2, which is a delete key operation. Observe that a dele eration as compensation commutes with insertion of a tuple with a different key, and encapsulates compos mpensation for the slot updating and index insertion. Compensation in this case is performed by logical un d hence the resulting execution is atomic (Theorem 1).

An entire class of applications for compensation (similar to the above example) can be found in the conte storage management in a database system. It is difficult to isolate the effects of an operation at the stora anagement level. Therefore, these effects are exposed to all the transactions. We list several specific examp

Chapter 3

Examples

this chapter, we present several examples to illustrate the various concept we have introduced so far. Throug
t this section we use the symbols T, CT, $dep(T)$, X, and S to denote a compensated-for transaction,
mpensating transaction, the corresponding set of dependent transactions, the execution of all these transa
ns, and the execution's initial state, respectively. A complete example that is based on an actual applicati
described in [SLKS91]. A short overview of this example is found in Section 7.1.

.1 Specification Example

e present a specification of what a generic CT should accomplish. Let $update(T, X)$ denote the set of databa
tities that were updated by T in execution X. The same notation is used for a set of transactions.

Definition 12. *Let $X(S) = S'$, and $X \equiv X' \circ X_{CT}$ (by Constraint 2). We specify CT, by characterizing*
all entities e:

$$S'(e) = \begin{cases} S(e) & \mathit{if} \quad e \notin update(dep(T), X) \\ (X'(S))(e) & \mathit{if} \quad e \in update(dep(T), X) \\ & \wedge \quad e \notin update(T, X) \\ X_{dep(T),e}(augment(S, X)) & \mathit{if} \quad e \in update(dep(T), X) \\ & \wedge \quad e \in update(T, X) \end{cases}$$

Observe that this specification conforms with Constraint 1. Before we proceed, we informally explain t
eaning of this type of compensation. If no dependent transaction updates an entity that T updates, CT undo
s updates on that entity. The value of entities that were updated only by dependent transactions is left inta
e value of entities updated by both T and its dependents should reflect only the dependents' updates as th
pear in X.

There is a certain subtlety in the second case of the definition which is illustrated next. Assume that
dated e. The modified e is read by a transaction in $dep(T)$ and the value read determines how this transacti
dates e'. After compensation, even though the initial value of e is restored (by the first case of the definitio
e indirect effect it had on e' is left intact (by the second case of the definition).

To further illustrate the type of compensation just described, we give a concrete example. Consider an airli
servation system with the entity *seats* that denotes the total number of seats in a particular flight, entity
at denotes the number of already reserved seats in that flight, and entity *reject* that counts the number
nsactions whose reservations for that flight have been rejected. Let **reserve(x)** be a simplified seat reservati
nsaction for x seats defined as:

serializable.

Example 8. Consider the set entities of Example 6, with the addition of a private entity u that belongs me transaction in $dep(T)$. Let the programs of T, $dep(T)$, CT, and the relation $\mathcal{R}$ be defined as follows:

$$\begin{array}{l} T \ = \ a := a+1, \ \ CT \ = \ a := a-1, \\ dep(T) = \{u := a; \ \ \textbf{if}\ u \geq 5\ \textbf{then}\ f(b) \ \ \textbf{else} \ \ g(b)\} \\ S' \ \mathcal{R}\ S'' \ \ iff \ \ (S'(b) \geq 0 \Rightarrow S''(b) \geq 0) \end{array}$$

en though $dep(T)$'s execution can branch differently when run alone and in the presence of T and CT, the t ferent executions produce final states that are related by $\mathcal{R}$.

.6 Compensation in Serializable Executions

e requirements from the dependent transactions in Theorems 2 and 3 are quite severe. Besides the mmutativity requirement imposed on the operations of the dependent transactions, there are restrictions shape of the programs (e.g., fixed or linear programs) in each of the theorems' premises. In both theoren ograms of dependent transactions are restricted to have no conditional statements. Clearly, in practice, the many transactions that do not stand up to any of these criteria. Reviewing the proofs of Theorems 2 a it is evident that the major obstacle is the lax restrictions on the interleaving of operations. As a matter t, only the EWSR assumption and Constraint 2, restrict concurrency in our exposition so far. In particul mma 2 indicates that operations of the forward, dependent and compensating transactions may interleave ferent orders for different entities. The almost arbitrary interleavings disallow treating a complete transacti a semantic unit. Thus, we are forced to build on the semantics of individual operations. The only way to re an entire transaction T was by the projected execution X_T. However, we have already noted that X_T is devo semantics and is just a syntactic derivative of X. Only if T is fixed or linear, X_T retains the semantics of T complete unit.

For these reasons, it is prudent to re-focus our attention on approximating atomicity by compensations ecutions that are serializable. Serializability allows one to treat entire transactions as if they are isolated. rticular, we can treat complete transaction programs as functions, rather than referring to individual ope ns. Moreover, we can capitalize on the $\mathcal{R}$-commutativity of entire transactions as functions. Consequent ializability gives us the leverage to deal with transaction programs with real control flow (i.e., conditio tements) and go beyond fixed and linear programs.

The only change in our notational machinery is the introduction of the transaction program as a function fr tes to states. The specifics of the control flow are abstracted by dealing with a program just as a function. the names of transactions, e.g., T, CT, and $dep(T)$ to denote these functions.

Theorem 4. *If CT $\mathcal{R}$-commutes with $dep(T)$, then every execution where T is a serialization point -atomic.*

proof. Let X be an execution where T is a serialization point, and let S be its initial state. Observe th ce both T and CT are serialization points, and $dep(T)$ is treated rather as a single (parent) transaction, X tually serializable.

$$\begin{array}{cl}
 & X(S) \\
= & \{\text{ both } T \text{ and } CT \text{ are serialization points }\} \\
 & (T \circ dep(T) \circ CT)(S) \\
= & \{\text{ function composition }\} \\
 & (dep(T) \circ CT)(T(S)) \\
\mathcal{R} & \{\ \mathcal{R}\text{-commutativity assumption }\} \\
 & (CT \circ dep(T))(T(S)) \\
= & \{\text{ constraint } 1 \ \} \\
 & (dep(T))(S) \\
= & \{\text{ Let } Y \text{ be that execution }\} \\
 & Y(S)
\end{array}$$

Theorem 4 is quite useful since it specifies a concurrency control policy that guarantees $\mathcal{R}$-atomicity. Name need to ensure that every potential compensated-for transaction be isolated (i.e., T is a serialization point) der to guarantee $\mathcal{R}$-atomicity in case of compensation. In the subsequent parts of this dissertation, we restr r attention to the use of compensation under the assumption of this theorem; namely, serializability is assume particular, the results reported in Chapters 6 and 7 deal with compensation in a distributed setting. The sults rely on Theorem 4 by assuming that at each individual site in the distributed system, the local executi

$$X =< \begin{array}{l} dep(T) : a := a+2,\ T : f(a),\ T : g(b), \\ dep(T) : g(b),\ CT : a := a+2,\ CT : b := b+10 > \end{array}$$

Observe that $X_{dep(T)}$ and X_{CT} do not commute but they do $\mathcal{R}$-commute for the given relation $\mathcal{R}$. Let t tial state be $S = \{a = 2,\ b = 15\}$. We have that $X(S) = \{a = 4,\ b = 15\}$, whereas $Y(S) = \{a = 4,\ b =$ d indeed X is *partially* $\mathcal{R}$-atomic.

Next, we relax the stringent requirement of having fixed programs.

Definition 10. *A program of a transaction is* <u>*linear*</u> *if it is a sequence of operations.*

ograms are sequences, but we allow operations to read multiple entities, that is, use local variables. Therefo ograms may not be fixed. An example for a linear transaction program is a program that gives a raise to ployees, where the raise based on some aggregated computation (for instance 10% of the minimum salary).

Definition 11. *Let $\mathcal{R}$ be a reflexive relation on augmented states. An operation f that updates e* <u>*preserves*</u>

$$(\forall e' \in adb : (S(e')\ \mathcal{R}_{e'}\ S'(e')) \Rightarrow ((f(S))(e)\ \mathcal{R}_e\ (f(S'))(e)))$$

Theorem 3. *Let X be an execution of $T, dep(T)$ and CT whose initial state is S. If the executions $X_{dep($ d X_{CT} $\mathcal{R}$-commute, X is EWSR, the programs of all transactions in $dep(T)$ are linear, $\mathcal{R}$ is transitive, a operations of $dep(T)$ preserve $\mathcal{R}$, then X is partially $\mathcal{R}$-atomic.*

Proof. Using the proof of Theorem 2 we can show that $(\forall e \in db : X_{dep(T),e}(S')\ \mathcal{R}_e\ (X(S))(e))$, where incides with S on the database state. Let Y be an execution of the transactions in $dep(T)$ that includes the sa erations as in $X_{dep(T)}$ and in the same order. Such a Y is a legitimate execution since $dep(T)$'s programs ear and have no conditional statements. Next, we show that $(\forall e \in db : Y_e(augment(S,Y))\ \mathcal{R}_e\ X_{dep(T),e}(S'$ ving the above two sets of $\mathcal{R}_e$ relations, we can apply the transitivity of $\mathcal{R}$ entity-wise and complete the pro

Since all programs in $dep(T)$ are linear we can treat Y merely as a sequence of operations, regardless of t uing transactions. Let $f_1 \circ \ldots \circ f_k$ be the sequence of all the operations of $dep(T)$ in the order of their appearan X (and hence also in Y).

We show that $(\forall e \in adb :\ Y_e(augment(S,Y))\ \mathcal{R}_e\ X_{dep(T),e}(S'))$, where S' coincides with S on the databa ate by induction on k.

<u>= 0</u>: $(\forall e \in adb : Y_e(augment(S,Y)) = S(e) = X_{dep(T),e}(S'))$.

<u>ductive step</u>: Let f_k update $e \in adb$. The final value of database entities other then e is computed by quence of at most $k-1$ operations. Therefore, we can apply the hypothesis of induction and get the followi $e' \in adb : (e' \neq e) \Rightarrow (Y_{e'}(augment(S,Y))\ \mathcal{R}_e\ X_{dep(T),e'}(S')))$. Let us focus on e itself. We can say th $(augment(S,Y)) = f_{n+1}(S'')$ and $X_{dep(T),e}(S') = f_{n+1}(S''')$ with the appropriate S'', and S'''. Since ea gument of f_k is computed using less than k operations in both X and Y, we can apply the hypothesis luction and get $(\forall e \in adb : S''(e)\ \mathcal{R}_e\ S'''(e))$. Since f_k preserves $\mathcal{R}$ we have completed the proof.

Example 7. Consider the set entities of Example 4, with the addition of a private entity u that belon some transaction in $dep(T)$. We use the relation $S'\ \mathcal{R}\ S''\ \ iff\ \ ((S'(b) \geq S'(a)) \Rightarrow (S''(b) \geq S''(a)))$. T ecution X is as follows:

$$X =< T : a := a+1,\ dep(T) : u := a,\ dep(T) : b := u+10,\ CT : a := a-1 >$$

oserve that X_{CT} and $X_{dep(T)}$ $\mathcal{R}$-commute (but do not commute), $dep(T)$ is linear (but not fixed), and X artially) $\mathcal{R}$-atomic.

Theorem 2. *Let X be an execution of $T, dep(T)$, and CT whose initial state is S. If the executions $X_{T'}$ a* $_{CT}$ *$\mathcal{R}$-commute for every transaction $T' \in dep(T)$, X is EWSR, and all programs of transactions in $dep(T)$ a ed, then X is partially $\mathcal{R}$-atomic.*

We first state a lemma that is used in several of the proofs.

Lemma 2. *Let X be an execution of $T, dep(T)$, and CT. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be some disjoint sets of transactic ch that $\mathcal{T}_1 \cup \mathcal{T}_2 = dep(t)$. If X is EWSR, then for all entities e:*

$$X_e \;\equiv\; X_{\mathcal{T}_1,e} \circ X_{T,e} \circ X_{\mathcal{T}_2,e} \circ X_{CT,e}$$

The proof of the lemma is a straightforward application of the assumption that X is EWSR along wi posing Constraint 2. We now turn to the proof of Theorem 2.

Proof. Let Y be an execution of the transactions in $dep(T)$ that includes the same operations as $_{lep(T)}$ and in the same order. Such a Y is a legitimate execution since $dep(T)$'s programs are fixed. First, serve that since the programs of transactions in $dep(T)$ use no private entities, for all states S and entities $_{lep(T),e}(S) = (X_{dep(T)}(S))(e) = (Y(S))(e)$. Since X_{CT} and $X_{T'}$ $\mathcal{R}$-commute for every $T' \in dep(T)$, then e definition of $\mathcal{R}_e$ and Lemma 1 we make a second observation:

$$\begin{array}{c}
(X_{CT,e} \circ X_{\mathcal{T},e})(augment(S, X_{CT} \circ X_{\mathcal{T}})) \\
\mathcal{R}_e \\
(X_{\mathcal{T},e} \circ X_{CT,e})(augment(S, X_{\mathcal{T}} \circ X_{CT}))
\end{array}$$

ere $\mathcal{T} \subseteq dep(T)$. We proceed as follows:

$$\begin{array}{ll}
 & (X(S))(e) \\
= & \{ \text{ Lemma 1 and Lemma 2 } \} \\
 & (X_{\mathcal{T}_2,e} \circ X_{CT,e})((X_{\mathcal{T}_1,e} \circ X_{T,e})(augment(S,X))) \\
\mathcal{R}_e & \{ \text{ second observation } \} \\
 & (X_{CT,e} \circ X_{\mathcal{T}_2,e})(augment((X_{\mathcal{T}_1,e} \circ X_{T,e})(augment(S,X)), X_{CT} \circ X_{\mathcal{T}_2})) \\
= & \{ \text{ private entities are partitioned and updated only once } \} \\
 & (X_{CT,e} \circ X_{\mathcal{T}_2,e})((X_{\mathcal{T}_1,e} \circ X_{T,e})(augment(augment(S,X), X_{CT} \circ X_{\mathcal{T}_2}))) \\
= & \{ \text{ Constraint 1 and programs of } dep(T) \text{ have no private entities } \} \\
 & X_{dep(T),e}(S) \\
= & \{ \text{ first observation } \} \\
 & (Y(S))(e)
\end{array}$$

hus, we have that for all entities e, $(Y(S))(e) \;\; \mathcal{R}_e \;\; (X(S))(e)$, and hence X is partially $\mathcal{R}$-atomic.

Example 6. Consider a database system with the following entities, parametric operations, and reflexi ation:

$$\begin{array}{rcl}
db & = & \{a : integer,\ b : integer\} \\
f(e) & :: & \textbf{if } e > 2 \;\; \textbf{then} \;\; e := e - 2 \\
g(e) & :: & \textbf{if } e > 10 \; \textbf{then} \;\; e := e - 10
\end{array}$$

$$\begin{array}{ll}
S' \,\mathcal{R}\, S'' & iff \\
 & (((S'(b) \geq 0 \wedge S'(a) \geq 10) \vee (S'(a) = 4)) \Rightarrow \\
 & ((S''(b) \geq 0 \wedge S''(a) \geq 10) \vee (S''(a) = 4)))
\end{array}$$

The predicates on a are present only to demonstrate the notion of partial $\mathcal{R}$-atomicity. The execution X follows (there is no need to give the program of $dep(T)$ since it is fixed):

Achieving even approximated atomicity is an intricate problem when the executions are non-serializable, allow them to be. The obstacle is, as mentioned before, that the programs of transactions in $dep(T)$ s ferent database states when T and CT are not executed, and therefore may generate an execution Y whi n be totally different than the original execution X. Hence, X and Y may not be related as required.

We state several theorems that formalize the interplay among the approximated atomicity notion, concurren ntrol constraints, restrictions on programs of dependent transactions, and commutativity. Each theorem llowed by a simplified example that serves to illustrate at least part of the theorem's premises and consequenc roughout this section, we assume that a compensating transaction complies with Constraints 1 and 2 of Secti

We start with definitions of weaker forms of commutativity and weaker forms of atomicity.

Definition 5. *Two functions from augmented states to augmented states, X and Y, commute with respect relation $\mathcal{R}$ on augmented states (in short, $\underline{\mathcal{R}\text{-commute}}$), if for all augmented states S, $(X \circ Y)(S)\ \mathcal{R}\ (Y \circ X)($*

oserve that when $\mathcal{R}$ is the equality relation we have regular commutativity.

Definition 6. *Let X be an execution of T, $dep(T)$, and CT whose initial state is S, and let $\mathcal{R}$ be a reflex lation on augmented states. The execution X is atomic with respect to R (in short $\underline{\mathcal{R}\text{-atomic}}$), if there exi execution Y of $dep(T)$ whose initial state is S such that $Y(S)\ \mathcal{R}\ X(S)$.*

oserve that regular atomicity is a special case of $\mathcal{R}$-atomicity when $\mathcal{R}$ is the equality relation. Since $\mathcal{R}$ lexive, the empty execution is always $\mathcal{R}$-atomic, regardless of the choice of R.

We motivate the above definitions by considering adequate relations $\mathcal{R}$ in the context of $\mathcal{R}$-commutativ d $\mathcal{R}$-atomicity. Let Q be a predicate on database states such that $O_{dep(T)} \Rightarrow Q$. Q can be regarded as eith consistency constraint, or a desired predicate that is established by $dep(T)$ (similarly to the predicate Q nstraint 3). Therefore, we would like to guarantee that compensation does not violate Q. Define $\mathcal{R}$ (in t ntext of X, Y and S) as follows:

$$Y(S)\ \mathcal{R}\ X(S) \ \ iff \ \ (Q(Y(S)) \Rightarrow Q(X(S)))$$

$\mathcal{R}$-atomic execution with such[2] $\mathcal{R}$ has the advantageous property that predicates like Q are not violated compensation. Such $\mathcal{R}$-atomic executions yield states that approximate states yielded by atomic executic the sense that both states satisfy some desirable predicates. In the examples that follow the theorems, we u ations $\mathcal{R}$ of that form.

Definition 7. *Let $S'\ \mathcal{R}\ S''$, and let a and b be values of entity e. We define a relation with respect to e, $\mathcal{R}$ e's values as follows: $a\ \mathcal{R}_e\ b \ \ iff \ \ ((S'(e) = a\ \wedge\ S''(e) = b)\ \vee a = b)$*

Definition 8. *Let X be an execution of T, $dep(T)$ and CT whose initial state is S, and let $\mathcal{R}$ be a reflex lation on augmented states. The execution X is $\underline{\text{partially } \mathcal{R}\text{-atomic}}$ if there exists an execution Y of $dep($ ose initial state is S such that $(Y(S))(e)\ \ \mathcal{R}_e\ \ (X(S))(e)$ for all database entities e.*

When an execution is partially $\mathcal{R}$-atomic, its final state can be partitioned as follows. For some entities, effects of T were completely removed, whereas for the rest of the entities, their values are related to the valu ey would have had, had T never been executed.

Definition 9. *A program of a transaction is $\underline{\text{fixed}}$ if it is a sequence of operations that use no private entit arguments.*

T's program is fixed then it has no conditional branches. Moreover, T cannot use local variables to store valu subsequent referencing. A sequence of operations, where each operation reads and updates a single databa tity (without storing values in local variables) is a fixed transaction.

[2] Since such a relation is anti-symmetric, we take care to always position the desired, hypothetical, execution (Y in this case), left-hand side, and the actual execution (X in this case) in the right-hand side of the relation.

mpensatory operations can be 'brought together', and then cancel each other's effects (by the enforcement onstraint 1), thereby ensuring atomic executions. The following theorem formalizes this idea.

Theorem 1. *Let X be an execution involving T, $dep(T)$ and CT. If each of the operations in $X_{dep(}$ mmutes with each of the operations in X_{CT}, then X is atomic.*

We illustrate this theorem by the following simple example:

Example 5. Let T_i, T_j and CT_i be a compensated-for transaction, a dependent transaction and t mpensating transaction, respectively. Let the programs of all these transactions include no condition statemen e., they are sequences of operations). We give an execution X, in which each operation is prefixed by the nar the issuing transaction.

$$X = < T_i : a := a + 2,\ \ T_j : u := b,\ \ T_j : a := a + u,\ \ CT_i : a := a - 2) >$$

early, every operation of T_j commutes with every operation of CT_i in X. Hence, X is atomic, and the executi at demonstrates atomicity is simply

$$Y \ = \ X_{T_j} \ = < T_j : u := b,\ T_j : a := a + u >$$

will become clear in Section 2.5, the fact that no condition statements appear in T_j is important.

Theorem 1 sets the stage for the use of logical undoing as the means for compensation. When applicab gical undoing allows exposing uncommitted updates early, yet ensures atomicity in case the updating transacti orts. These benefits, however, can be attained only when the undo operations commute with the operations e dependent transactions as prescribed in Theorem 1. We do not elaborate any further on logical undoing as s already been studied thoroughly (e.g, refer to [BSW88, WHBM90, MHL$^+$90, ML89]. One point we would li point out, however, concerns the perspective we advocate regarding logical undo. Typically, commutativi d logical undo are mentioned as means to enhance concurrency. Our point of view slightly shifts the emphas e underline the ability to logically undo an externalized transaction, yet retain atomicity without resorting scading aborts.

Our main emphasis in this chapter is on more liberal forms of atomicity by compensation, where the resu executing the dependent transactions in isolation may be different from their results in the presence of t mpensated-for, and the compensating transactions. One way of characterizing these weaker forms of atomici by qualifying the set of entities for which the equality in Definition 4 (atomicity definition) holds. In Section 3 define a type of compensating transaction that ensures atomicity with respect to a certain subset of entiti ır main contribution, however, focuses on other weak forms of atomicity that approximate in a semantic ser re atomicity.

.5 Approximating Atomicity

this section we introduce weak forms of atomicity by compensation, where the results of an execution th cludes compensation only *approximate* the results of executing the dependent transactions in isolation.

Let us denote the execution of transactions T, $dep(T)$, and CT as X, and the execution without compensatio ., an execution of only $dep(T)$, as Y. In an approximated form of atomicity, the final state of X is only *rela* the final state of Y.

The relation should serve to constrain CT, and prevent it from violating consistency constraints and oth sirable predicates established by $dep(T)$. Thus, the relation should enforce some 'goodness' properties, f stance: "if a consistency constraint predicate holds on the final state of Y, it should also hold on the final sta X."

crucial since T's effects are undone by CT, and hence, predicates established by T and preserved by $dep(T)$ t persist after the compensation. It is the responsibility of whoever defines CT to enforce Constraint 3.

Constraints 1 and 2 will be assumed to hold for all compensating transactions, hereafter. Constraint 3, whi more intricate and captures more of the semantics of compensation, will be discussed further in Section 2.5

.4 Atomicity by Compensation

r some applications, it is acceptable that an execution of the dependent transaction, without the compensate r and the compensating transactions, would produce different results than those produced by the executi th the compensation. On the other hand, other applications might forbid compensation unless the outcome ese two executions is the same. Next, we make explicit the above criterion that distinguishes among types mpensation by defining the notion of atomicity by compensation.

Definition 4. *Let X be the execution of T, CT, and $dep(T)$ whose initial state is S. Let Y be some executi only the transactions in $dep(T)$ whose initial state is also S. The execution X is atomic by compensation (ort, atomic), if $X(S) = Y(S)$.*

The execution Y can be any execution of $dep(T)$. As far as the definition goes, different sets of (sub)transactic $dep(T)$ may commit in X and in Y, and conflicting operations may be ordered differently. The key point at $X(S) = Y(S)$. If an execution is atomic then compensation does not disturb the outcome of the depende nsactions. The database state after compensation is the same as the state after an execution of only t pendent transactions in $dep(T)$. All direct and indirect effects of the compensated-for transaction, T, ha en erased by the compensation.

Transactions in $dep(T)$ see different database states when T and CT are not executed, and therefore genera execution Y which can be totally different than the execution X. This distinction between the executions d Y, which is the essence of the important notion of atomicity by compensation, would not have been possil d we viewed a transaction merely as sequence of operations rather than a program.

A delicate point arises with regard to atomicity when S does not satisfy $I_{dep(T)}$. Such situations may occ en T establishes $I_{dep(T)}$ for $dep(T)$ in such a manner that $dep(T)$ *must* follow T in any execution. Hence, is compensated-for, there is no execution of $dep(T)$, Y, that can satisfy the atomicity requirement. We mod ch situations by postulating that if $I_{dep(T)}(S)$ does not hold, then $Y(S)$ results in a special state (the *undefin* ate) that is not equal to any other state and hence X is indeed not atomic.

Example 4. We illustrate Definition 4 by considering the following two executions over read and wr erations (the notation $r_i[e]$ denotes reading e by T_i, and similarly $w_i[e]$ for write, and c_i for commit):

$$W = < w_j[e] , r_i[e] , c_j , c_i >$$
$$Z = < w_j[e] , r_i[e] , w_i[e'] , c_i >$$

e execution W is recoverable [BHG87]. History Z is not recoverable. If however, CT_j is defined, T_j can still orted. Let us extend Z with the operations of CT_j and call the extended execution Z'. Z' is atomic provid at Z'_{T_i} would have been generated by T_i's program, and the same value would have been written to e', had n in isolation starting with the same initial state as in Z'.

The key notion in the context of compensation, as we defined it, is *commutativity* of compensating operatic th operations of dependent transactions. Significant attention has been devoted to the effects of commutati erations on concurrency control [Kor83, Wei88, BR87, Reu82]. Our work parallels these results as it explo mmutativity with respect to recovery. In all of our theorems we prefer to impose commutativity requirements T rather than on T, since CT is less exposed to users, and hence constraining it, rather than constraining T, eferable. Predicated on commutativity, the operations of the compensated-for transaction and the correspondi

- *T_j reads e after T_i has updated e.*
- *T_i does not abort before T_j reads e.*
- *Every transaction (if any) that updates e between the time T_i updates e and T_j reads e, is aborted before reads e.*

The above definition is adapted from the definition of "reads-from" of [BHG87].

The key point is that admitting executions that do not avoid cascading aborts and supporting the undo mmitted transactions is predicated on the existence of the compensatory mechanisms needed to handle undoi ternalized transactions. In the sequel, T denotes a compensated-for transaction, CT denotes the correspondi mpensating transaction, and $dep(T)$ denotes a set of transactions that depend on T. This set of depende nsactions can be regarded as a set of related (sub)transactions that perform some coherent task.

Constraint 1. *For all executions X, if $X_{T,e} \circ X_{CT,e}$ is a contiguous subsequence of X_e, then $(X_{T,e} \circ X_{CT,e})$ where I is the identity mapping.*

The simplest interpretation of Constraint 1 is that for all entities e that were updated by T but read by her transaction (since $X_{CT,e}$ follows $X_{T,e}$ immediately in the execution), CT amounts simply to undoing nsequently, if there are no transactions that depend on T, (i.e., no transaction reads T's updated data entitie en CT is just the traditional *undo*(T). The fact that CT does not always just undo T is crucial, since the effe compensation depend on the span of execution from the execution of the compensated-for transaction till n initiation. If such a span exists, and T has dependent transactions, the effects of compensation may va d can be very different from undoing T. The fact that compensation degenerates to simple undo as specifi Constraint 1, is used later in the dissertation. In Chapter 7, traditional undo is modeled by a compensati nsaction.

There are certain operations that cannot be undone, or ev mpensated-for. In [Gra81] these type of operations are termed *real* (e.g., dispensing money, firing a m e). Constraint 1 does not apply for these type of operations. For simplicity, we do not discuss compensati real operations in this chapter. We defer discussion of non-compensatable operations to Section 6.5.

Definition 3. *A transaction T is a serialization point in an execution X if $X \equiv X' \circ X_T \circ X''$, such th transaction has operations both in X' and in X''.*

Constraint 2. *A compensating transaction must be a serialization point.*

This constraint is referred to as *isolation of recoveries* and it plays a key role later in the dissertation. It asse at a transaction should either see a database state affected by T (and not by CT), or see a state following C mination. More precisely, transactions should not have operations that conflict with CT's operations schedul th before and after CT's operations, or in between CT's first and last operations. It is the responsibility e concurrency control protocol to implement this constraint. This constraint is elaborated later on in t ssertation and protocols for enforcing it are devised (see Section 5.3).

In what follows, we use the notation O_T and I_T to denote the output and input predicate of transacti respectively. The same notation is used for a set of transactions. These predicates are predicates over t tabase state.

Constraint 3. *Let Q be a predicate defined over the database state, if $(O_{dep(T)} \Rightarrow Q) \wedge (I_T \Rightarrow Q)$ th $_{CT} \Rightarrow Q$.*

onstraint 3 is appropriate when Q is a either general consistency constraint, or a specific predicate that ablished by $dep(T)$ (that is, one of the collective tasks of the transactions in $dep(T)$ was to make Q tru ormally, this constraint says that if Q was established by $dep(T)$, and is not violated by undoing T (sin $\Rightarrow Q$), then it should be preserved by CT. Observe that the assumption that Q holds initially (i.e., $I_T \Rightarrow$

Proof. Let $X = X' \circ f_k$, where $X' = f_1 \circ \ldots \circ f_{k-1}$. The proof is by induction on k. Some of the proofs s dissertation are in 'Dijkstra' style. Namely, each step in the proof is explained by a hint within .

$\underline{= 0}$:

$$\begin{array}{cl} & (X(S))(e) \\ = & \{\ X \equiv I\ \} \\ & S(e) \\ = & \{\ \text{definition 1}\ \} \\ & (augment(S, I))(e) \\ = & \{\ X \equiv I\ \} \\ & X_e(augment(S, X)) \end{array}$$

<u>ductive step</u>: Observe that $X(augment(S, X \circ X')) = X(augment(S, X))$. This holds since private entities a dated only once, and are used only after being updated. Therefore, updating private entities in X' is irreleva the execution X in this case.

$$\begin{array}{cl} & X_e(augment(S, X)) \\ = & \{\ \text{projection}\ \} \\ & ((X_e)(augment(S, X))(e) \\ = & \{\ \text{definition of } X\ \} \\ & [f_k((X'_e)(augment(S, X)))](e) \\ = & \{\ \text{observation above}\ \} \\ & [f_k((X'_e)(augment(S, X')))](e) \\ = & \{\ \text{hypothesis of induction, entity-wise}\ \} \\ & [f_k(X'(S))](e) \\ = & \{\ \text{function composition}\ \} \\ & (X(S))(e) \end{array}$$

In our discussions we consider the following types of executions:

- A execution X is *serial* if for every two transactions T_i and T_j that appear in X, either all operations of appear before all operations of T_j or vice versa.

- A execution X is *serializable* (SR) if there exists a serial execution Y such that $X \equiv Y$.

- A execution X is *entity-wise serializable* (EWSR) if for every entity e there exists a serial execution Y su that $X_e \equiv Y_e$.

we shall see shortly, EWSR executions are going to be quite useful in our work. We impose very we nstraints on concurrent executions in order to exclude as few executions as possible from consideration.

.3 Specification Constraints

ith the aid of the tools developed in the last section, we are in a position to define compensation more formal though compensation is an application-dependent activity, there are certain guidelines to which every compe ing transaction must adhere. After introducing some notation and conventions we present three specificati nstraints for defining compensating transactions. These constraints provide a very broad framework for defini ncrete compensating transactions for concrete applications, and can be thought of as a generic specification f compensating transactions.

Definition 2. *A transaction T_j* depends *on transaction T_i in an execution if there exists an entity e su at the following conditions hold:*

A key notion in the treatment of compensation is *commutativity*. We say that two sequences of operations, d Y, *commute*, if $(X \circ Y) \equiv (Y \circ X)$. Two operations *conflict* if they do not commute. Observe that defini erations as functions, regardless to whether they read or update the database, leads to a very simple definiti the key concept of commutativity. Compare our definition to those of [Wei88, BR87] for example.

Part of the orderings implied by the total order in which operations are composed to form an execution a bitrary, since only conflicting operations must be totally ordered. In essence, our equivalence notion, wh stricted to database state, is similar to final-state equivalence [Pap86]. However, in what follows, we shall ne equate executions that are not necessarily over the same set of transactions, which is in contrast to final-sta uivalence (and actually to all familiar equivalence notions).

We denote by X_T ($X_{\mathcal{T}}$) the sequence of operations of a transaction T (a set of transactions $\mathcal{T}$) in an executi involving possibly other transactions. A *projection* of an execution X on an entity e is a subsequence of at consists of the operations in X that updated e. We denote the projection of X on e as X_e. The sa tation is used for a projection on a set of entities. When X_T is projected on entity e the resulting sequence noted $X_{T,e}$.

It should be noted that X_T does not reflect the general control structure of the program of T since it is just t quence of T's operations appearing in a particular X. In essence, X_T is a curried function whose dependen the particular interleaving in X and the particular initial state is embedded in it. The remaining arguments $_T$ are the arguments of its operations. These arguments can still be assigned values that are different from t lues they were assigned in X itself. For instance, refer to Example 2. In $Z_{T_1}(S)$, u is assigned a different val an in $Z(S)$. This peculiarity is not relevant to our results, since we concentrate in Section 2.5 on transactio th fixed control structure (i.e., no conditional statements). Hence, in Section 2.6, when we focus on arbitra nsactions, we do not deal with sequences like X_T any longer.

When a projection on an entity is applied to a state, we are interested in the resulting value of that particul tity. Therefore, we use $X_e(S)$, as a shorthand for $(X_e(S))(e)$.

The astute reader may have noticed that $X_e(S)$ is not well defined, and in particular it is not necessarily equ $(X(S))(e)$. Since X_e includes only operations that update e, and since private entities are updated only on e value of all private entities is undefined when executions are projected on database entities. To rectify th omaly we define the function *augment* from database states and executions to augmented states as follows.

Definition 1. *Let S be a database state, and X an execution, then:*

$$(augment(S,X))(e) = \begin{cases} S(e) & \text{if } e \in db \\ (X(S))(e) & \text{if } e \in adb - db \end{cases}$$

In essence, the function *augment*, assigns private entities the values they hold after X was applied to herefore, when an execution X is projected on a database entity, by applying it to $augment(X,S)$ rather th st to S, we avoid the anomaly. We illustrate the function *augment* and its use in the following example:

Example 3. Let $u \in (adb - db)$, and $\{a, b\} \subseteq db$. Consider the following execution $X =< u$ $b := f(u) >$. Let $S(u) = \emptyset$ (undefined value), and $S(a) = 1$. Then, $X_b(S) = f(S(u)) = f(\emptyset)$, where $(S))(b) = f(1)$. However, $(augment(S,X))(u) = (X(S))(u) = 1$, and then indeed $X_b(augment(S,X)) = f($

Essentially, the augmented state $augment(S,X)$ represents the *view* [Pap86] operations have on the databa execution X applied to state S.

Lemma 1. *For all executions X*

$$(\forall e \in adb : \ (X(S))(e) = X_e(augment(S,X)))$$

2.2 Executions and Correctness

e use the framework for alternative correctness criteria set forth in [KS88]. Explicit *input* and *output predica* er the database state are associated with transactions. The input predicate is a pre-condition of transacti ecution and must hold on the state that the transaction reads. The output condition is a post-conditi ich the transaction guarantees on the database state at the end of the transaction provided that there is ncurrency and the database state seen by the transaction satisfies the input condition. Thus, as in the standa odel, transactions are assumed to be generated by correct programs, and responsibility for correct concurre ecution lies with the concurrency control protocol.

Observe that the input and output predicates are excellent means for capturing the semantics of a databa stem. We use the convention that predicates (and hence semantics) can be associated with a set of transactio milarly to the way predicates are associated with nested transactions in [KS88]. That is, a set of transactions pposed to collectively establish some desirable property, or complete a coherent task. This convention is mo eful in domains where a set of subtransactions are assigned a single complex task.

We do not elaborate on the generation of interleaved or concurrent executions of sets of transaction progran ce this is not central to understanding our results. However, the notion of an execution, the result of tl erleaving, is a central concept in our model. A *execution* is a sequence of operations, defining both a total orc nong the operations, as well as a function from augmented states to augmented states that is the functior mposition of the operations. We use the notation $X =< f_1, \ldots, f_n >$ to denote an execution X in whi eration f_i precedes f_{i+1}, $1 \leq i < n$. Alternatively, we use the functional composition symbol '$\circ$' to compo erations as functions. That is, $X = f_1 \circ \ldots \circ f_n$ denotes the function from augmented states to augment ites defined by the same execution X. We use the upper case letters at the end of the alphabet, e.g., $X, Y,$ denote both the sequence and the function an execution defines.

The equivalence symbol '$\equiv$' is used to denote equality of executions as functions. That is, if X and Y a ecutions, then $X \equiv Y$ means that for all augmented states S, $X(S) = Y(S)$. Observe that since executic d operations alike are functions, the function composition symbol '$\circ$' is used to compose executions as well erations.

When a (concurrent) execution of a set of transaction programs A is initiated on a state S and generates ecution X, we say that X is a *execution of* A whose *initial state* is S.

Example 2. Consider the transaction program T_1 of Example 1. Since T_1 has a conditional stateme ere are two possible executions, X and Y, which can be generated when T_1 is executed in isolation. We list t ecutions as sequences of operations:

$$\begin{aligned} X &= < u := a, v := b, c := f(c,v) > , \\ Y &= < u := a, v := b, w := c, b := g(u,w) > \end{aligned}$$

t $S = \{ a = 1 , b = 0 , c = 2 \}$ be database state, then S is an initial state for X. $X(S) = S'$, whe $(c) = f(2,0)$. Consider a *concurrent* execution of T_1 and T_2 of the previous example. We show two (out of t ny possible) executions, Z and W, whose initial state is S given above. Each operation is prefixed with t me of the transaction that issued it.

$$\begin{aligned} Z =< \; & T_2 : a := 0, \; T_1 : u := a, \; T_2 : b := 1, \\ & T_1 : v := b, \; T_1 : w := c, \; T_1 : b := g(u,w) > \\ W =< \; & T_2 : a := 0, \; T_2 : b := 1, \; T_1 : u := a, \\ & T_1 : v := b, \; T_1 : w := c, \; T_1 : b := g(u,w) > \end{aligned}$$

oserve that $Z(S) = W(S) = S''$, where $S'' = \{ a = 0 , b = g(0,2) , c = 2 \}$. Observe that $Z \equiv W$.

Entities in our scheme can be of arbitrary granularity and complexity. Examples for entities are pages of da d index files, or abstract data types like stacks and queues. Accordingly, sample reading operations are re page, stack top, is-empty queue, and sample updating operations are write a page, stack push and pop, a sertion into a queue. Notice that the above sample reading operations only read the database state witho dating it. On the other hand, a blind write only updates the database state but does not read it. Final suming integer-type entities, an increment operation both reads and updates an integer entity.

We are in a position now to introduce the notion of a transaction as a program. A *transaction program* is quence of *program statements*, each of which is either:

- An operation.
- A *conditional statement* of the form:

  $$\textbf{if}\ \ b\ \textbf{then}\ \ SS1\ \textbf{else}\ SS2$$

  where $SS1$ and $SS2$ are sequences of program statements, and b is a predicate that mentions only priva entities and constants.

e impose the the following restrictions on the operations that are specified in the statements:

- The set of private entities is *partitioned* among the transaction programs. An operation in a program cann read nor update a private entity that is not in its own partition;
- private entities are updated only once;
- An operation reads a private entity only after another operation has updated that entity.

nese restrictions are for the sake of convenience in proofs and they do not restrict the expressibility of the mod

Example 1. Consider the following sets of entities: $db = \{a, b, c\}$, and $adb = db \cup \{u, v, w\}$, and the followi o transaction programs, T_1 and T_2:

```
T1:  begin
            u:=a;
            v:=b;
            if u > v then c:= f(c,v)
                     else  begin
                                w:=c;
                                b:= g(u,w)
                           end
      end

T2:   begin
             a:=0;
             b:=1
      end
```

oserve that operation `f` both updates and reads entity c. T_2 illustrates operations that read no entities.

ce originally transactions read data items updated by T and acted accordingly, whereas now T's operations ha nished but its indirect impact on its dependent transactions is still apparent. The only formal way to examine mpensated execution is by comparing it to a hypothetical execution of only the dependent transactions, witho e compensated-for transaction. We use the comparison of the compensated execution with the hypotheti ecution that does not include the compensated-for transaction, as a key criterion in our exposition. Generati is hypothetical execution and studying it requires the introduction of the *transactions' programs* which a erefore, indispensable for our purposes.

A *transaction program* can be defined in any high-level programming language. Programs have local (i. ivate) variables. In order to support the private (i.e., non-database) state space of programs we define t ncept of an *augmented state.* The augmented state space is the database state space unioned with the priva ate spaces of the transactions' programs. The provision of an augmented state allows one to treat reading a dating the database state in a similar manner. Reading the database state is translated to an update of t gmented state, thereby modeling the storage of the value read in a local variable.

Thus, a *database*, denoted as db, is a set of data *entities.* The *augmented database*, denoted as adb, is a set tities that is a superset of the database; that is, $db \subset adb$. An entity in the set $(adb - db)$ is called a *priv tity.* Entities have identifying *names* and corresponding *values.* A *state* is a mapping of entity names to enti lues. We distinguish between the *database state* and the state of the augmented database, which is referred the *augmented state.* We use the notation $S(e)$, to denote the value of entity e in a state S. The symbols d e (and their primed versions, S', e', etc.) are used, hereafter, to denote a state and an entity, respectively.

Another deviation from the classical transaction model is the use of semantically-richer operations instead e primitive read and write. Having such operations allows refining the notion of conflicting versus commutati erations [BR87, Wei88]. That is, it is possible to examine whether two operations commute and hence can ecuted concurrently. By contrast, in the classical model, there is not much scope for such considerations sin write operation conflicts with any other operation on the same entity.

An *operation* is a function from augmented states to augmented states that is restricted as follows:

- It updates at most one entity (either a private or a database entity);
- it reads at most one *database* entity, but it may read an arbitrary number of private entities;
- it can both update and read only the same database entity.

e use the shorthand notation $e_0 := f(e_1, \ldots, e_k)$ to denote a single operation f. We say that f *updates* enti , and *reads* entities $e_1, \ldots, e_k$. The *arguments* of an operation are all the entities it reads. There are two spec rmination operations, *commit*, and *abort*, that have no effect on the augmented state. Operations are assum be executed *atomically.*

It is implicitly assumed that all the arguments of an operation are meaningful; that is, a change in their val use a change in the value computed by the operation. The operations in our model reconcile two contradicto als. On the one hand, operations are functions from augmented states to augmented states, thereby giving t xibility to define complex and semantically-rich operations. On the other hand, the mappings are restrict that at most one database entity is accessed in the same operation, thereby making it feasible to all omic execution of an operation. Although only one database entity may be accessed by an operation, any local variables (i.e., private entities) as needed may be used as arguments for the mapping associated wi e operation. Having private entities as arguments to operations adds more semantics to operations. Havi nctions for operations allows us to conveniently compose operations by functional composition, thereby maki quences of operations functions too.

eserving some sense of atomicity. Initiating a compensating transaction is caused by a decision to abort t rward transaction. In order to maintain (at least relaxed) atomicity, we claim this decision to be non-reversib d make sure it is robust to failures of all sorts.

There are other special characteristics. A compensating transaction does not exist by its own right; it is alwa garded within the context of the forward transaction, and it is always executed after the forward transaction. mpensating transaction is driven by a program that is a derivative of the program of the forward transactio e binding of forward and compensating transactions is explicit, and is realized as the compensating transacti ts as input a trace of the execution of the forward transaction (in the form of the latter's log records, ample).

A mundane example taken from "real life" exemplifies some of the characteristics of compensation. Conside tabase system that deals with transactions that represent purchasing of goods. Consider the act of a custom urning goods after they have been sold. The compensated-for transaction in that case is a particular purcha d the compensating transaction encompasses the activity caused by the cancellation of the purchase. T mpensating transaction is bound to the compensated-for transaction by the details of the particular sale (e. ice, method of payment, date of purchase). The effects of purchasing transaction might have been externaliz different ways. For instance, it might have triggered a dependent transaction that issued an order to t pplier in an attempt to replenish the inventory of the sold goods. Furthermore, the customer might have be ded to the store's mailing list as a result of that particular sale. The actual compensation depends on t evant policy. For example, the customer may be given store credit, or full refund. Whether to cancel the ord om the supplier and whether to retain the customer in the mailing list are other application-dependent issu th which the compensating transaction must deal.

.2 A Transaction Model

the classical transaction model [Pap86, BHG87] a transaction is viewed as a sequence[1] of read and wr erations that map consistent database states to consistent states when executed in isolation. A concurre ecution of a set of transactions is represented as an interleaved sequence of read and write operations, and d to be *serializable* if it is equivalent to a serial (non-concurrent) execution. *Serializability* is the correctne terion of this model.

This approach poses severe limitations on the use of compensation. First, sequences of uninterpreted rea d writes are of little use when the semantically-rich activity of compensation is considered. Second, the u serializability as the correctness criterion for applications that demand interaction and cooperation amo ssibly long-duration transactions was questioned by the work on concurrency control in [KS88, KKB88, FO8 nce we target compensation as a recovery mechanism for these kind of applications, our model does not rely ializability as the only correctness notion.

2.1 Transactions and Programs

transaction is a sequence of operations that are generated as a result of the execution of some program [Gra8 e exact sequence that the program generates depends on the database state "seen" by the program. In t ssical transaction model only the sequences are dealt with, whereas the programs are abstracted and are tle use. Given a concurrent execution of a set of transactions (i.e., an interleaved sequence of operation mpensation for one of the transactions, T, can be modeled as an attempt to cancel the operations of T wh ving the rest of the sequence intact. The validity of what remains from that execution is now in serious dou

[1] We use a sequence which implies a total order, only for the ease of exposition. One can regard the sequence to be conflict-equival HG87] to a partial order of operations.

verse operations are executed in the reverse order of their execution.

It is instructive for our purposes to evaluate how the two traditional undo methods affect concurrency. I
ysical method is used, data items that were updated by T may be neither read nor written by other transactio
til T commits or aborts. This requirement is known as *strictness* [BHG87]. By restricting concurrency in t
anner it is made possible to undo T by simply restoring the physical before images of the relevant data iten
te that no operations may access the data items updated by T from the point they are affected to the point th
e committed or recovered. Consequently, compensation cannot rely on physical undo methods. The strictne
quirement can be lifted for certain operations if a logical method is used, and thus enhance concurrency. Name
two operations commute, they can be executed concurrently, regardless of whether their issuing transactio
e committed or not. If one of these transactions must be undone, the corresponding inverse operation can st
ncel the effect of the aborted operation. A common example in the context of commutative operations a
crement and *decrement* operations which commute with each other and among themselves [Reu82]. A da
m can be incremented concurrently by two uncommitted transactions. If one of the transactions aborts,
ect can be undone by decrementing the item appropriately, leaving the effects of the other increment inta
cause of the commutativity of the operations, the logical undo yields a state that is identical to the state th
uld have been reached had the forward transaction never executed. Note that (only) commutative operatio
y access the data items updated by T from the point they are affected to the point they are committed
covered.

Logical undo is based on the semantics of the operations. A decrement operation is recognized as the inver
increment only since the semantics of both is known. Likewise, compensation is a semantically-rich recove
thod. However, it is a generalization of logical undo that is applicable even for non-commutative operatio
nce it is not based on commutativity, compensation does not guarantee the undoing of all the direct a
direct effects of the forward transaction. In particular, some of the effects of the dependent transactions, whi
e indirect effects of T, may remain intact. Compensation does guarantee, however, that a *consistent* state
ablished based on semantic information. We emphasize that unlike logical undo, the state of the databa
er compensation took place may only approximate the state that would have been reached, had the forwa
nsaction never executed. In spite of the differences, compensation is still a method for automatically undoi
nsactions, just like the traditional methods. Compensation, however, is applicable in the more general ca
ere the undone transaction has already exposed its updates.

We propose the notion of *compensating transactions* as the vehicle for carrying out compensation. We u
e notation CT to denote the compensating transaction specific to the forward transaction T. A compensati
nsaction possesses the fundamental properties of a transaction along with some special characteristics.
pears atomic to concurrently executing transactions (that is, transactions do not observe partially compensat
tes); it conforms to consistency constraints; and its effects are durable. However, a compensating transaction
ery special type of transaction. Under certain circumstances, it is required to *restore* consistency, rather th
rely preserve it. Also, compensating transactions have a unique failure atomicity requirement which is explain
xt. Compensating transactions cannot voluntarily abort; the choice to either abort or to commit is prese
ly for the forward transaction. A compensating transaction offers the ability to reverse this choice, but t
pability to abort the compensation is not supported. Moreover, the underlying implementation should ascerta
at once compensation is initiated, it will eventually complete. Namely, compensating transactions should n
subject to a system-initiated abort. Also, their completion should be guaranteed despite system crashes
her resuming them from a save-point, or retrying them. Finally, a compensating transaction must be design
avoid a logical error leading to abort. This stringent requirement is referred to as *persistence of compensati*
d is recognized in [GMS87, GM83, Vei89, GMGK$^+$90, Reu89]. We elaborate on the mechanisms needed
plement persistence of compensation in Section 4.1.4. The rationale behind persistence of compensation

Chapter 2

Single-Transaction Compensation

ı informal overview of most of the features of compensation is given in Section 2.1. In Section 2.2, we present mantically-rich and flexible transaction model. The formal results concerning compensation comprise the re the chapter. It should be noticed that Sections 2.5 and 2.6 serve as the basis for material presented later e dissertation.

.1 Overview of Compensation

ıe most common method for obliterating the effects of an aborted transaction T, is to maintain a recove ; and provide the *undo*(T) operation which restores the state of data items updated by T to the value th d just prior to the execution of T. The undo operation removes all the direct effects of T on the databa owever, if some other transaction has read data values written by T, undoing T is not sufficient. The indire ects of T must be removed by aborting the transactions that have read T's updates, and thus are affect its execution. Aborting the affected transaction may trigger further aborts. This undesirable phenomenc lled *cascading aborts*, can result in uncontrollably many transactions being forced to abort because some oth ınsaction happened to abort.

The purpose of *compensation* is to handle situations where it is desired to undo a transaction T whc committed updates have been exposed. Undoing T, however, should not trigger aborting other transactic at read the exposed updates; that is, cascading aborts should be avoided. We refer to T as the *compensated-f forward* transaction. The set of transactions that are affected by (reading) the data values written by T a ferred to as *dependent transactions* (of T), and are denoted *dep*(T). Compensation faces the intricate task doing the forward transaction while obliterating the effects of the dependent transactions to a minimal exter d preserving data consistency. Only with the aid of the specific semantics of the application at hand can tl sk be accomplished. Intuitively, compensating for T can be though of as performing (an approximation of) t verse of the function performed by T.

To understand compensation better, we compare and contrast it with the traditional methods of transacti do. There is a dichotomy of the traditional methods into physical (or state-based), and logical metho R83, MHL^{+}90]. Using physical methods, before a data item is updated by a transaction T, its physical ima stored. These images are typically saved on a log, and are referred to as *before images*. If a transaction abor e before images of all the data items it has updated are reinstated, thereby restoring the state of the databa ior to T's execution. In contrast, logical methods are based on having inverse operations associated with t erations of transactions. The execution of a forward transaction is recorded on a log, too, however descriptic d parameters of operations are stored rather than before images. To undo a transaction, the correspondi

r compensating transactions in a recovery management subsystem are highlighted in Chapter 4. In the latt
rt of this chapter we sketch a design methodology for compensating transactions. The transition from sing
nsaction compensation to full-fledge semantics-based recovery of composite transaction in distributed syster
made in Chapter 5. The problem of obtaining transaction atomicity in a distributed system is explained in th
apter. The concept of *isolation of recovery* which is a backbone of the latter part of the dissertation is informa
esented there, too. Chapters 6 and 7 present two specific methods for solving the problem of atomicity in
stributed system [LKS91b, LKS91a]. The common denominators and the differences of these two methods a
derlined in Section 5.6. We review related work, and sketch future research directions in Chapters 8 and
spectively. The dissertation concludes in Chapter 10.

omicity is guaranteed as the effects of a transaction that is finally aborted are undone semantically by mpensating transaction. Relaxing standard atomicity interacts in a subtle way with correctness and concurren ntrol issues. Accordingly, a correctness criterion that incorporates the isolation property, is proposed. T rrectness criterion reduces to serializability when no global transactions are aborted, and excludes unacceptal ecutions when global transactions do fail. We devise a family of practical protocols that ensure this correctne tion. The results on relaxed atomicity are of particular importance for multidatabases, where the local autonor the integrated systems cannot be compromised.

In summary, the salient contributions of this dissertation are:

- Introducing compensation as the viable solution to the recovery needs of long-duration and cooperati transactions.
- Specification of formal criteria for the proper use of compensation as a recovery paradigm.
- Applying semantic recovery in distributed databases and analyzing the ramifications.
- Trading standard atomicity for relaxed atomicity, and consequently coming up with pragmatic methods a protocols that alleviate the inherent difficulties associated with commitment in distributed systems.

.2 Structure of the Dissertation

ypically a long-duration or a distributed transaction is decomposed into subtransactions [GM83, GMS87, CP8 ereby introducing a nested, or multi-level transaction hierarchy [Mos87, BSW88]. This hierarchical layeri as found useful for cooperative environments as well [KKB88, KLMP84]. The decomposition is often logic at is, a subtransaction is associated with a coherent unit of work. For a distributed transaction, however, t composition can be more arbitrary, as all the actions executed at a single site are defined as a subtransactio ne resultant transaction hierarchy introduces spheres of atomicity, since the subtransactions as well as the ro ansaction possess atomicity properties. It is instructive to cast the intuitive notion of early exposure of updat thin this well-structured framework. The commit of a subtransaction is an early externalization of updat om the root transaction point of view.[1] Thus, if a root transaction is to abort, its committed subtransactio ose which have completed their task and exposed their updates, should be compensated-for. In summary, ing the hierarchical structuring, exposing uncommitted data translates to committing a subtransaction pri the commit of the root transaction. Accordingly, compensation is applied on a subtransaction basis.

This hierarchical structuring guides a bottom-up structuring of this dissertation. We start with a buildi ock of a single subtransaction and investigate how a committed subtransaction can be compensated-for. e course of this exposition, a subtransaction is treated as an independent transactional unit with comple mantics, and the encompassing hierarchy is entirely disregarded. For instance, the transaction model present Section 2.1, defines a model for a single (sub)transaction. Only in Chapters 5 and 6, we step one level up a roduce composite transactions again, mainly in a distributed context. The results obtained earlier are used vance the study of semantic recovery in the broader scope of composite transactions.

The remainder of this dissertation is organized as follows. Chapter 2 lays the foundation for the thesis by givi rigorous basis to compensation. In particular, the infringement upon standard atomicity when compensati employed is pinpointed. The material for this chapter is largely from [KLS90a]. Chapter 3 illustrates t ndamental concepts presented earlier by a set of examples. Practical issues concerning the support need

[1] The reader should regard this analogy in the context of a single-level nested transaction, where visibility of updates of a su nsaction is not restricted to only sibling subtransactions, but is rather not restricted at all. The model we have in mind is akin gas [GMS87] (also known as open nesting), more than to the original nested transactions of Moss [Mos87] (also known as clo sting).

ration and distributed transaction management, and would provide critical functionality for enterprises bas cooperative transactions. The thesis defended in this dissertation focuses on *semantics-based recovery* as t quisite method. Semantics-based recovery has two dual facets, *compensation* and *retry*. The duality of the t mantic recovery methods is rooted in the traditional undo/redo paradigms [BHG87].

Semantic undoing, referred to as *compensation*, is carried out by a *compensating transaction* which is associat th a specific *forward transaction*. A compensating transaction faces the intricate task of undoing its forwa nsaction while obliterating the effects of other transactions to a minimal extent and preserving data consisten nly with the aid of the specific semantics of the application at hand can this task be accomplished. Ideal mpensation can be thought of as performing the inverse of the function associated with the forward transactio owever, compensating for a transaction does not guarantee the physical undoing of all the direct and indire ects of the forward transaction. That is, the state of the database after compensation has taken place m ly approximate the state that would have been reached had the forward and compensating transactions nev ecuted. We formally identify conditions for ensuring that executions with compensations approximate in ceptable way ideal executions. This formal basis sets forth general requirements from compensating transactio d shapes a methodology for their design.

Aborting, or compensating-for, a transaction that encompasses elaborate human activity (e.g., a long-runni nsaction in a collaborative design environment [KKB88]), or intensive and costly computation (e.g., a lon nning data processing transaction [GMS87]) is often counter-productive. Instead, it is preferable to identify t use for the failure and act accordingly with the objective of saving the work associated with such a transactio at is, under certain circumstances, forward rather than backward recovery is desirable. We refer to t tivities associated with the forward recovery of a failed transaction as *retry*. Retry ranges from tradition do to automatic execution of code, failure diagnostics and exception handling. Similarly to compensatio ry depends on the semantics of the application at hand. In this dissertation we concentrate on compensati hapters 2,3, and 4) and cover retry rather briefly (Section 5.7). Since the two methods have much in comm d because of their duality, one might expect the results for compensation to carry over for retry, however th sumption requires further research. Our results do not rely in any manner on the specifics of retry, howev ey are amplified once such a method is assumed. We acknowledge that there are actions that are neith mpensatable nor retriable. The ideas and protocol we devise are such that they can accommodate transactio turing a blend of actions, some of which are not semantically recoverable.

Supporting atomicity of multi-site transactions in a distributed system is equated with the loss of the lo tonomy of the individual sites, and the problems of long-duration delays and blocking. The two-phase comm PC) protocol [Gra78] embodies these deficiencies. These hard problems can be alleviated by employing seman covery, and by trading standard all-or-nothing atomicity for a weaker notion of *relaxed atomicity*. Facing t evant impossibility results in distributed computing, this new direction is well justified. Relaxed atomicity aracterized by an asynchronous process of recovery from decentralized and uncoordinated local decisions as hether to commit or abort a multi-site transaction. This recovery process finally leads to a unanimous outcom e to the asynchrony introduced to the commit procedure, non-atomic executions of transactions occur, a need to isolate them from other transactions until they are recovered. A formal model that unifies the t al methods of semantic recovery, namely compensation and retry, is constructed. In this model, an *isolati operty* is defined, and a protocol that satisfies this property is presented.

Based on the notion of relaxed atomicity, we devise a transaction management protocol that combines tw ase locking [BHG87] with a variant of 2PC. The protocol is based on the optimistic assumption that in mo ses a transaction that reaches its *lock point* [BHG87] (i.e., the point where the transaction has already acquir its locks), will indeed commit. Employing this optimistic protocol, locks may be released early under certa cumstances, thereby avoiding the maladies of the standard 2PC protocol. Relaxed, rather than standar

Chapter 1

ntroduction

e motivation and a synopsis of the thesis defended in this dissertation is summarized in Section 1.1. Secti
2 introduces the components and structure of this work, and gives a brief overview of each chapter.

.1 Semantics-Based Recovery: Motivation and Thesis

e cornerstone of the transaction paradigm is the notion of *atomicity*. Transaction atomicity asserts that
nsaction either completes entirely and *commits* its effects, or *aborts* and has no visible effect on the databa
e principle that forms the basis for obtaining transaction atomicity in most contemporary database syste
to allow transactions to access only committed data; data that has been updated by transactions that ha
eady committed. That is, a transaction that requests to access data items affected by another transaction,
layed until the other transaction is committed or aborted. There is a large range of database environments f
ich this standard approach to transaction atomicity is excessively restrictive and even not appropriate. V
ghlight the prominent problems below:

- When transactions are of long duration, the delays caused by waiting for their termination are prolong
  accordingly. For short transactions executing concurrently with a long-lived transaction, such delays impa
  response time by orders of magnitude, and are thus intolerable [Gra81].

- In a variety of applications, the transaction paradigm is used to model collaborative activities [KLMP8
  HR87, KKB88, RM89]. In order to promote the cooperative nature of these activities there is a need
  exchange, and thereby expose, uncommitted data objects among transactions.

- In distributed database systems, atomicity of multi-site transactions is achieved by employing an aton
  commit protocol that coordinates among the sites participating in the execution of the transaction. Exposi
  updated data to other transactions only after this protocol terminates translates to severe, and actua
  unbounded, delays in transaction processing. In particular, if the processing of the transaction at each s
  is of different duration, the coordinated commit causes lengthy delays unnecessarily.

- Multidatabases are a specific type of distributed database system where several database systems are i
  tegrated to enable the processing of multi-site transactions [hdb90]. Enforcing atomicity strictly in su
  integrated environments compromises the distinctive and crucial property of *autonomy* of the individu
  systems.

A method that allows exposing uncommitted data, yet preserves transaction atomicity without induci
scading aborts is thus highly desirable. Such a method would alleviate performance problems related to lor

Contents

Semantics-Based Recovery in Transaction Management Systems[1]

Eliezer Levy
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188 USA
levy@cs.utexas.edu
Fax (512) 471-7028

Chairpersons of the Supervisory Committee:
Professor Abraham Silberschatz
Associate Professor Henry F. Korth

[1] This report is a revision of Levy's Ph.D. dissertation. The research was funded by the National Science Foundati rants IRI-8805215 and IRI-9003341) and by the IBM and NEC Corporations. Levy was supported by a Unisys Gradua lowship.

SEMANTICS-BASED RECOVERY IN TRANSACTION MANAGEMENT SYSTEMS

Eliezer Levy

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

TR-91-29 August 1991

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712