Recoverable Persistent Memory for SmartCard

Didier Donsez[1], Gilles Grimaud[2], and Sylvain Lecomte[2]

[1] Laboratoire d'Informatique et de Mathématiques Appliquées de Valenciennes
Université de Valenciennes, Le Mont Houy BP 311
59304 Valenciennes Cedex France,
E-mail: donsez@univ-valenciennes.fr
[2] RD2P - Recherche et Développement Dossier Portable
CHR Calmette - rue du Pr. J. Leclercq
59037 Lille Cédex - France
Tel: +333 20 44 60 46 - Fax: +333 20 44 60 45
E-mail: grimaud@lifl.fr, lecomte@lifl.fr

Abstract. Smartcard is well adapted to store confidential data and to provide secure services in a mobile and distributed environment. But many cases of smartcard application failure can corrupt data in smartcard persistent memory. In this paper, we propose a recoverable persistent memory to maintain data consistency in a smartcard. Then, we adapt and compare two recovery algorithms used in Database Management Systems (shadow paging and before-image logging) to the smartcard memory features. At last, we present a prototype, which demonstrates the feasibility of these algorithms in a smartcard.

Keywords. Recovery, Persistent Memory, Transaction, Smartcard

1 Problems of Failure in SmartCard

Smartcard is well adapted to store confidential data and to provide secure services in a mobile and distributed environment[Cor96]. So, the range of their applications is continuously growing (electronic commerce, medicine, phone card, etc). These applications are more and more designed as distributed applications [Lec97][Lec98]. The computation of the application is spread over several computers on a network. The smartcard is one of these computers when it is slotted into a terminal.

However, as in distributed systems, failures should have been considered. We distinguish the node failure in which a node (smartcard or other) crash and the communication failure between nodes. In the smartcard context, the communication failure is mainly the failure of the link between the smartcard and the other nodes of the distributed application. This can be caused by the failure of the terminal and by the wireless link when the smartcard is plugged in a mobile terminal such as cellular phone. The second kind of failure is the node crash and especially the smartcard crash. Since the card user (a human being) can

unplugged the card from the terminal before the application completion and the terminal supply electricity power to the card, the smartcard is suddenly powered off and all the RAM contents is suddenly lose. A second cause of smartcard crash is an execution error due to a bad argument or to a programmation error in the application.

Communication and node failures can introduce incoherence in the smartcard (and in the other nodes) data. Smartcard can complete the computation (for example to credit an electronic purse) and valid updates of its data after the communication failure even if the other nodes decided to discard their part of the computation (for example to debit a bank account). The smartcard crash may cause also an incoherence in the data since a part of updates may be in RAM and lost by the crash.

The developer should have to handle these problems. The communication failure can be resolve by distributed validation protocols such as the two-phase commit protocol [Gra93]. A simple way to manage the incoherence of the data during a smartcard crash is to restore the previous state of data, i.e. the state of the data at the beginning of the computation.

In this paper, we do not study the problem of distributed application, but only the recovery failure of the data in case of a smartcard crash. The recovery is based on the transactional execution model briefly present in section 2. Section 3 describes the adaptation of two recovery algorithms to the smartcard memory context. Section 4 presents briefly the demonstrator and section 5 concludes.

2 Transactional Model and Recovery Persistent Memory

A simple way to manage failure is to make atomic all updates done by an application (distributed or centralized). Atomicity of updates done by a group of code lines is a part of a larger concept, the concept of transaction [Gra93]. A transaction is a group of actions (i.e. code lines). The transaction completes in two ways: the *Commit* in which all effects (i.e. updates) is durable either the *Abort* in which all effects are discarded. So, the transaction guaranties the ACID properties (Atomicity, Coherence, Isolation and Durability). Atomicity means that all or none of the modifications are validated. Consistency says that a transaction takes data in a coherent state and returns it in a coherent state. Isolation isolates the effect on the transaction until the commit so concurrent transactions can not seen the uncommitted updates. Durability enforces that effects of the transaction are accessible even in case of failure (media...).

In the smartcard context, the natural group of actions for which Atomicity and Durability must be guarantied is the APDU command (Application Protocol Data Unit)[Iso95]. But the transactional model can be used to execute atomicaly a group of actions over several APDU commands or several cards sessions (i.e. the card is plugged several times in different terminals before the commitment of a transaction) [Lec97]. On the other hand, a command APDU may be composed of several transactions.

New models and platforms for application development hide the difference between volatile (RAM) and non-volatile memories (Hard Disks...) by supplying Recoverable Persistent Memory (RPM)[Atk83][Sat93]. RPM provides mechanisms to allow the atomicity, since it can undo updates or make them persistent at commit time. Many works have focused on RPM in Operating Systems (OS) and in Database Management Systems (DBMS) [Sin92][Whi94]. The main techniques are described by [Ber87]. These techniques suppose that the non-volatile memories are magnetic media such as hard disks or tapes to guaranty the properties of Atomicity and Durability.

To store persistently data, DBMS and OS consider mainly block devices such as hard disks or tapes to make persistent data. These media can not be addressed directly by the CPU so data are loaded by bulk in the buffers in RAM. However some recovery algorithms [Cop89][Tec97] consider special hardware such as UPS-RAM (Uninterruptible Power Supply RAM) to improve the performance of logging operations by maintaining safe the logging buffers.

3 Recovery Manager in SmartCard

Smartcard persistent memories provide different features than magnetic media. Indeed, smartcard manufacturers use mainly the 3 following technologies to implant persistent memory in SmartCard: EEPROM, FeRAM, and FlashRAM. With these technologies, CPU can address (i.e. read and write) directly memory cells.

But in case of Flash-RAM, a quite long erasing operation (100 ms) must precede writing operation. So the Memory Manager (MM) must emulate a paginated memory with a page size close to 8 to 16 words. MM imposes to write in non-volatile memory only through a RAM cache[Bel66] so it translates logical addresses of data toward RAM or non-volatile memory addresses. But this needs a translation table residing in RAM and sometimes in non-volatile memory. Caching in RAM extends also smartcard's life duration since writing operations damage EEPROM media (manufacturers guaranty only 10000 writing operations in an EEPROM cells)[Cas95].

Taking into account these features, we adapt two recovery algorithms designed for Database Management Systems to the smartcard persistent recoverable memory manager. The two algorithms are the "Shadow Pages" algorithm and the " Before-Image Logging " algorithm. The following sections present efficient and simple RPM over EEPROM and FlashRAM technologies. Nevertheless, we do not consider the After-Images Logging which is very equivalent to the Shadow page algorithm in the smartcard context.

3.1 Adaptation of the Shadow Pages Algorithm

The shadow page algorithm preserves the original state of data. Each write operation into a page is performed on a copy of this page, named "shadow page". In order of that, the algorithm uses a translation table. With FlashRAM,

this table is loaded in RAM from non-volatile memory at the beginning of the transaction. All writing operations are performed in shadow pages allocated from the list of free pages. The translation table maps toward the shadow pages. To commit atomically all updates, the translation table is written in a shadow copy. So the previous state of the translation table is preserve. If the failure occurs before the commit phase of the transaction, the original table is reloaded so current updates are discarded. If the failure occurs during the commit phase of the transaction, the MM reloads the latest persistent copy of the table which is not corrupted (i.e. all write operations of the table are completed or not before the failure). With EEPROM, the translation table stays in persistent memory. To map the shadow page of an update page, the translation table is updated in a shadow copy. Shadow page algorithm is shown in fig. 1.

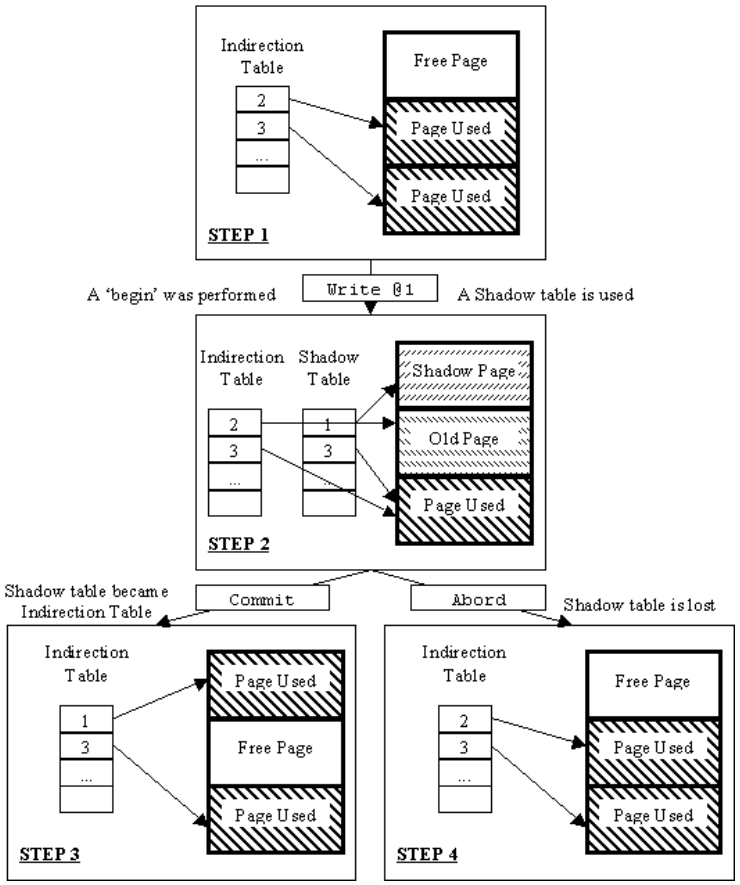

Fig. 1. Shadow Pages Machanism

This algorithm is easy to implant but it has major drawbacks. Firstly, only one transaction is running at a time (i.e. batch processing model) because the translation table can not be shared between several simultaneous transactions. Secondly, with FlashRAM, the translation table takes a great part of the RAM so this reduces the cache performance.

3.2 Adaptation of the Before-Image Logging Algorithm

This algorithm does not need a persistent translation table. It uses only a small in-RAM table (12 bytes) for the needs of the caching mechanism (if it is enabled). The Before-Image Logging algorithm keeps the original data values in a distinct area of the non-volatile memory called Before-Image Log. The log is a file that contains log records. Before performing a writing operation, the original value is saved in a new log record. The log record contains also an incremental sequence number, the value identifier (i.e. the address of a page if whole pages are logging or the offset and the size in case of sub-page logging). If a failure occurs before and during the commit phase, the MM plays forward the log to replace the new (uncommitted) values by the original (committed) values saved in the log. With EEPROM technology, sub-page logging is possible (if the caching mechanism is enable). Sub-page logging consist in saving only the updated parts of the page in a log record. Indeed before each cache replacement, the updated page (which is in the RAM cache) is compared with the non-volatile page to detect updated words and to log these original corresponding values. The operation called "Diffing" is used for Client-Server OO-DBMS Recovery [Sin92][Whi95]. With Flash-RAM technology, the Diffing allows only whole page logging since erasing operation deletes the state of a whole page before writing.Before image log file is shown in fig 2.

The Before-Image Logging presents several advantages. Firstly, no huge translation table is required. Therefore, cache size is maximal and this improves performances by reducing the number of cache replacements. This reduces also the number of log records if the Diffing is used. Secondly, several transactions can access concurrently to the persistent memory. Thirdly, the MM can easily rollback over multiple checkpoints and can provide recovery mechanisms to implement extended transaction models [Elm92] such as nested-transactions [Mos85]. However, the two former features require a complex (for the smartcard) recycling (compression) of the log.

4 Implementation of a Prototype

We develop a demonstrator to evaluate performances of these recovery algorithms (Shadow Pages, Before-Image Logging with and without Diffing) for the two following non-volatile memory : EEPROM and FlashRAM. These algorithms are benchmarked with several configuration of memory (varying size of RAM and varying size of persistent memory) and several application workloads. The presentation of these benchmarks and these results are not presented in this paper.

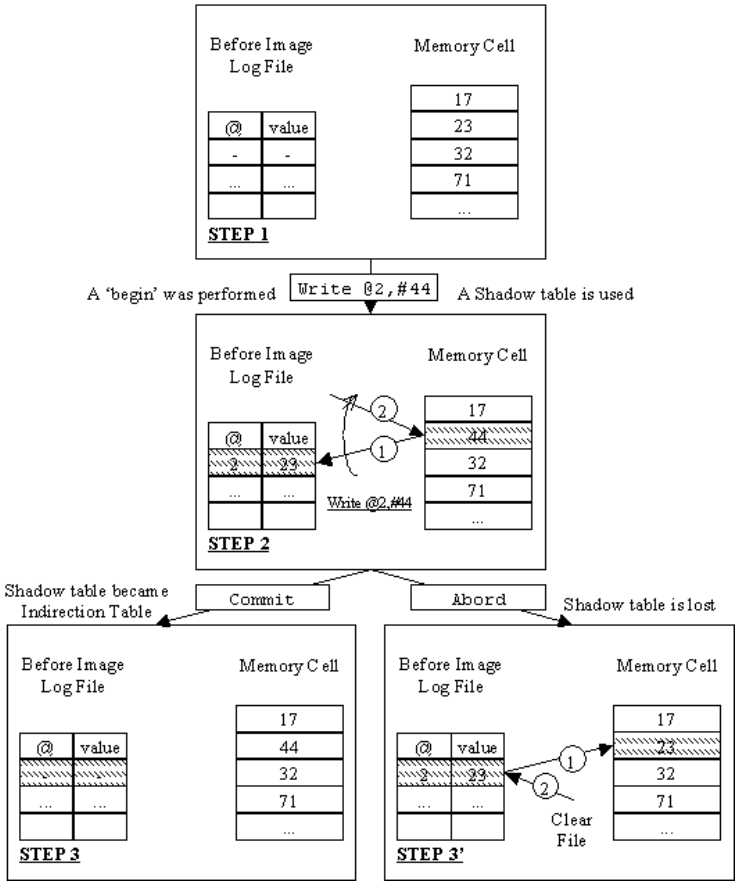

Fig. 2. Before image log file

A part of the demonstrator is also integrated on GemXpresso, the GEMPLUS's JavaCard [SUN97] compliant smartcard. This card is also used in a distributed platform to participate in the secure completion of distributed applications.

5 Conclusion and Perspectives

In this paper, we first present the failures, which can make incoherent data stored in a smartcard. Then to recover these failures, we propose mechanisms based on the transactional model. We propose to implement Recoverable Persistent Memory in smartcards. RPM hide the difference between volatile (RAM) and non-volatile memories (FlashRAM, EEPROM, ...) and guaranty the atomicity and durability of a group of code lines. We adapt and implement DBMS recovery algorithms to the smartcard memory context. These algorithms are the " Shadow Pages " one and the " Before-Image Logging " one.

However, smartcards will be completely integrated when they will be able to participate to distributed transactions[Lec98]. This assumes that ACID properties are guaranties by every node of the distributed transaction. The ACID properties in distributed transactions oblige the card to participate to a distributed validation protocol such as the two phases commit protocol specified by X/Open and OMG. In another hand, the smartcard can execute several transactions simultaneity. The Isolation property must be guaranty by the card. We work on concurrency control mechanisms such as the two-phase locking which provides a coherent sharing of data.

References

[Atk83] Malcom Atkinson, Ken Chishlom, Paul Cockshott, Richard Marshall, " Algorithms for a Persistent Heap ", Software, Practice and Experience, 13(3) :259-271, March 1983.

[Bel66] Belady, "A study of Replacement Algorithms for Virtual Storage Computers", IBM Systems Journal 5, 1966, pp78-101.

[Ber87] P.A. Bernstein, V. Hadzilacos, N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison-Wesley, Reading , MA, 1987.

[Cas95] Esprit program EP8670, " CASCADE: Operating System ", European project ESP8670 , April 1995.

[Cor96] V. Cordonnier, " The future of SmartCards : Technology and Application ", In proceedings IFIP World Conference on Mobile Communication, Camberra, September 1996.

[Elm92] A.K. Elmagarmid, " Database Transaction Models for Advanced Applications ", Morgan Kaufmann Publishers, 1992

[Gra93] Jim Gray, Andreas Reuter, " Transaction Processing : Concepts and Technics ", Morgan Kaufmann Publishers, 1993.

[Iso95] International Standard Organisation (ISO), " Identification cards, integrated circuit cards with contacts : Part 4 inter-industry command for interchange ", International Standard ISO/IEC 7816-4, 1995

[Lec97] Sylvain Lecomte, Didier Donsez, "Intégration d'un Gestionnaire de Transaction dans les cartes à microprocesseur " , Proceeding of NOTERE, Nouvelles Technologies de la Répartition, Pau, France, 4-6 Novembre 1997, pp. 347-362

[Lec98] Sylvain Lecomte, " COST-STIC : Cartes Orientées Services Transactionnels et Systèmes Transactionnels Intégrant des Cartes",PhD Dissertation, University of Lille I, Villeneuve d'Ascq, France, November 1998.

[Mos85] J.E.B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing", Boston, MIT Press, 1985.

[Tec97] Wee Teck Ng, Peter M. Chen, "Integrating Reliable Memory in Databases", VLDB'97, Proceedings of 23th International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece. Morgan Kaufmann 1997, ISBN 1-55860-470-7, pp76-85.

[Cop89] George P. Copeland, Tom Keller, Ravi Krishnamurthy, Marc Smith, "The Case For Safe RAM", VLDB 1989, pp327-335

[Sat93] M. Satyanarayanan, H.H . Mashburn, P. Kumar, D.C. Steere, J.J. Kistler, " Lightweight Recoverable Virtual Memory ", Proc of 1993 Symposium on Operating System Principles, pp 146-160, December 1993.

[Sin92] V. Singhal, S.V. Kaddak, P.R. Wilson, "Texas: An Efficient, Portable Persistent Store", Proc. of the Fifth Intl Workshop on Persistent Object System, San Miniato, Italy, September 1992.

[SUN97] Sun Microsystems, "JavaCard 2.0 Language subset and Virtual Machine Specification", Rev 1.0 final, October 1997.

[Whi94] S.J. White, D.J. DeWitt, "QuickStore : A High Performance Mapped Object Store", Proc. of the 1994 ACM SIGMOD Conf. Minneapolis, Minnnesota, May 1994.

[Whi95] Seth J. White, David J. DeWitt "Implementing Crash Recovery in QuickStore: A Performance Study" In proceedings of the 1995 ACM SIGMOD Conference, pp 187-198