PicoDBMS: Validation and Experience

Nicolas Anciaux*, Christophe Bobineau*, Luc Bouganim*,
Philippe Pucheral*, Patrick Valduriez**

* PRISM Laboratory
Univ. of Versailles, France
<Firstname.Lastname>@prism.uvsq.fr

** Lip6 Laboratory
Univ. of Paris 6, France
<Firstname.Lastname>@lip6.fr

1. Introduction

Smartcard is the most secure and cheap portable computing device today and became the world's highest-volume market for semiconductors in the year 2K. As smartcards become multi-application (by hosting a dedicated Java Virtual Machine) and more and more powerful (32 bit processor, more than 1MB of stable storage [Gem99]), the need for database management arises. Embedding database management (query processing, access rights, transaction control) in the card simplifies and makes application code smaller and safer.

Smartcard DBMS cover the need of confidential, portable and sharable databases such as: (i) diplomatic, military or business data, (2) user profiles (passwords, cookies, bookmarks, licenses, agenda…), and (3) personal folders (scholastic, healthcare, loyalty…).

However, smartcards have severe hardware limitations (tiny RAM, very costly write in stable storage…) making traditional database techniques irrelevant. Small footprint DBMS (e.g., Oracle 8i Lite), designed for portable computers and PDA, do not address the more severe limitations of smartcards.

In a recent paper [BBP+00], we addressed this problem and proposed the design of what we call a PicoDBMS. Since then, we have implemented a full-fledged PicoDBMS prototype in JavaCard. We are porting it on Bull's 32 bit smartcard, which we got only recently. Thus, the objective of the demonstration is to:

- Validate our design by building a complex database application (with respect to the smartcard) on our prototype and showing the benefits of the approach.
- Validate our techniques by showing that they match the smartcard hardware constraints.

**Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001**

2. The Health Card Demonstrator

The proposed demonstration is a sample of healthcare application which is representative of personal folders and has strong database requirements. The information stored in the future health cards should include the holder's identification, insurance data, emergency data, the holder's doctors, prescriptions and even links to heavier data (X-ray examination…) stored on hospital servers. Different users may share data in the holder's folder: the doctors who consult the patient's past records and prescribe drugs, the pharmacists who deliver drugs, the insurance agents who refund the patient, public organizations which maintain statistics or study the impact of drugs correlation in population samples and the holder herself.

The demonstration will show that our prototype meets the healthcard application's requirements, that is: (i) being able to manage a significant amount of data (more in terms of cardinality than in terms of volume), (ii) supporting complex queries (*e.g.,* a doctor asks for the sum of antibiotics prescribed to the patient in the last three months), (iii) handling sophisticated access rights using views and aggregate functions (*e.g.,* a statistical organization may be allowed to access aggregate values only but not the raw data). The benefit of hosting the healthcare data on smartcards rather than on a server is high availability and high privacy of medical records.

3. PicoDBMS Design and Implementation

Smartcard's hardware security makes it the ideal storage support for private data. The impact of security is that the data as well as the query engine, the view manager and the access right manager must be confined in the chip. Only functions that do not impact the query result can be externalized to the terminal (e.g., the GUI, the query parsing…). When designing the PicoDBMS's embedded components, we must follow several design rules derived from the smartcard's properties [BBP+00]:

- *Compactness rule:* minimize the size of data structures and the PicoDBMS footprint.
- *RAM rule:* minimize the RAM usage given its

extremely limited size.

- *Write rule*: minimize writes in stable storage given their dramatic cost (≈1 ms/word).
- *Read rule*: take advantage of the fast read operations (≈100 ns/word).
- *CPU rule:* take advantage of the overabundant CPU power (i.e., the time complexity of an algorithm is not a major concern).
- *Access rule:* take advantage of the low granularity and direct access capability of the stable memory.
- *Security rule:* never externalize private data from the chip.

2.1 PicoDBMS storage model

From the read and access rules, it follows that a PicoDBMS is a typical main-memory DBMS. Thus, it can take advantage of a pointer-based storage and access model to meet compactness and performance altogether. The PicoDBMS exploits a combination of *Flat Storage (FS)*, *Domain Storage (DS)* and *Ring Storage (RS)*. By storing the attribute values embedded in the tuples, FS is adequate when the attribute does not present value redundancy. DS precludes any duplicate value to occur by grouping values in domains and by using references in place of the attribute values. RS addresses index compactness. RS links together all tuples sharing the same attribute value through a circular chain of pointer headed by this value. This chain of pointer is stored again in place of the attribute values, providing a similar and compact implementation of both select and join indices.

2.1 Query processing

Traditional query processing uses main memory or disk for storing intermediate results. The RAM and write rules preclude the use of these algorithms in our context. In addition, the security rule precludes any intermediate results to be externalized outside of the chip.

To conform to these three rules, the PicoDBMS execution model must be able to tackle any query, whatever be its complexity and the volume of data it involves, with - almost - no RAM consumption. To this end, we proposed a new execution strategy called extreme right deep tree which executes all operators (including select, join, aggregate and duplicate removal) in a pure pipeline fashion [BBP+00].

4. PicoDBMS Demonstration

The demonstration platform (Figure 1) includes a full-fledged PicoDBMS prototype, a JDBC driver, a user interface and a monitor interface. Our PicoDBMS prototype is written in JavaCard 2.1 and runs today on a smartcard simulator and on the most recent Bull's 32 bit smartcard. It includes a storage manager, a query manager, an access right manager and a view manager. The PicoDBMS footprint is around 30KB.

The user interface comes as a Java applet that can be downloaded on any Java-enabled terminal. It allows formulating SQL queries, which are sent to the PicoDBMS via a dedicated JDBC driver. The monitor interface delivers both static information (EEPROM image) and runtime information (RAM usage, number of read and write operations, execution time, query execution plans) about the PicoDBMS.

One difficulty in assessing the performance of our PicoDBMS is that its internal state cannot be easily sniffed when embedded in a chip. Thus, we use the smartcard simulator to experiment with the PicoDBMS on databases larger than the smartcard storage capacity and help understanding the behavior of the PicoDBMS' internals, in combination with the monitor interface.

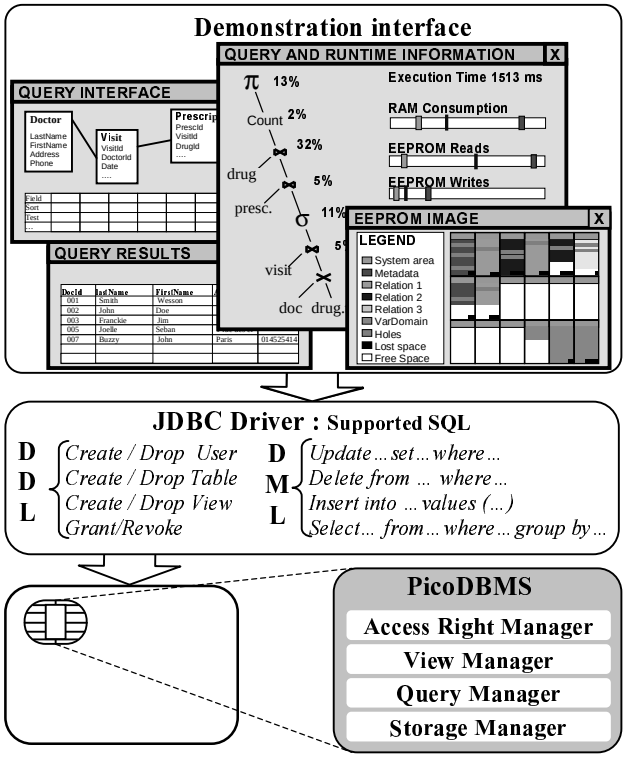

Figure 1: *Demonstration Platform*

Acknowledgments

We would like to thank Pascal Urien for the technical support provided on the Schlumberger / Bull-CP8 smartcard platform, and Nacim Rahali and the students of the ISTY engineering school for helping us implementing the demonstration.

References

[BBP+00] C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez. PicoDBMS : Scaling Down Database Techniques for the Smartcard (Best Paper Award). *Int. Conf. on Very Large Databases (VLDB)*, 2000.

[Gem99] Gemplus. SIM Cards: From Kilobytes to Megabytes.www.gemplus.fr/about/pressroom/, 1999.