Archiving of Data in an EXPRESS/SDAI Database

Axel Herbst*
IBM Heidelberg Scientific Center
Vangerowstr. 18
69115 Heidelberg
Germany

e-mail: aherbst@vnet.ibm.com

Abstract

Even though EXPRESS is primarily an information modeling language, database implementations are arising which accept EXPRESS as a DDL (Data Definition Language). Data manipulation is preferably done via SDAI (Standard Data Access Interface). We illustrate our database prototype implementing the C++ Binding of SDAI atop the object-oriented database system ObjectStore. Furthermore, we address the increasing demand for the long-term retention of mass data by an appropriate archiving concept. We point out how to integrate an archiving service into SDAI. Thus SDAI applications are enabled to indicate which data can afford e.g. longer access times for the sake of (tertiary) storage economy. Contrary to usual database archiving, this selection is based on the data model, namely EXPRESS. Necessary SDAI extensions include a select and an archive operation as well as the introduction of archive repositories.

Key words

EXPRESS, SDAI, STEP databases, object-oriented databases, archiving

*also: University of Kaiserslautern, Department of Computer Science

1 EXPRESSing more than information models

It is widely acknowledged that EXPRESS is well-suited for its originally intended purpose, which is information modeling [SW94]. Thousands of lines of EXPRESS code have been written for many parts of the STEP standard [Owe93]. EXPRESS models can also be found beyond the product data domain. Every EUG conference includes reports on "non STEP" applications such as the management of genomic data [SB93] or the use of EXPRESS in an information infrastructure for research in High Energy Physics [DHSV94]. Regardless of the specific application area, there is a growing interest not only in modeling and exchanging data but also in storing and sharing it. Consequently, **database systems** have been implemented which provide an EXPRESS view of the managed data. EXPRESS has not yet been adopted as the native Data Definition Language (DDL) of a database system which otherwise provides the crucial features such as concurrency control, query processing, or recovery support. Rather, EXPRESS constructs are mapped e.g. to SQL table definitions or – only recently – to persistent classes of an object-oriented database system (ODBS) [LH94].

To offer a truly comprehensive EXPRESS view, applications need to access and manipulate the data in terms of *entity instance* and *attribute value* rather than in terms of *tuple* or *data member*. Hence, the SDAI (Standard Data Access Interface [ISO94]) specification plays the role of the EXPRESS database's Data Manipulation Language (DML). A persistent SDAI implementation requires a corresponding mapping of SDAI operations to those operations provided by the database system. With some restrictions, this mapping can be straightforward if the C++ Binding of SDAI is used and if C++ is seamlessly supported by the database system. This is the case with our prototype, which exploits the ODBS *ObjectStore*[1] [LLOW91].

We consider our EXPRESS/SDAI database an extensible research vehicle to study and implement partial solutions to the following problem: especially EXPRESS/SDAI databases have to cope with a continous increase in the amount of data being stored. This is because data such as product configurations or simulation results is rarely deleted but frequently added to a database. Product data, in particular, needs to be kept over long periods of time for economic and legal reasons. Retention times of years and decades exacerbate the problem. Furthermore, the data must be stored at a reasonable cost. Our approach is to **archive** selected data appropriately. In particular, an archiving component of a database system must "understand" EXPRESS. An archiving facility needs to be callable by an SDAI application program. In other words, the possibility to archive data should be an **integrated SDAI service**.

The remainder of this paper is organized as follows. Section 2 describes our database prototype. Section 3 elaborates on the novel archiving concept and presents the necessary extensions of SDAI. Our ongoing and future work is indicated in section 4.

[1] ObjectStore is a trademark of Object Design, Inc.

2 An EXPRESS/SDAI database prototype

2.1 Implementation architecture

Among the three basic alternatives

1. mapping of EXPRESS alone, native DML of the employed database system,
2. implementation of an SDAI software layer atop a database system, or
3. from-scratch design of a dedicated, full-fledged SDAI database system,

we chose the second option. More precisely, we implemented the C++ Binding of SDAI close to the specification "N350" [Pri93]. As sketched in figure 1, four components are involved in creating an EXPRESS database and running SDAI applications.

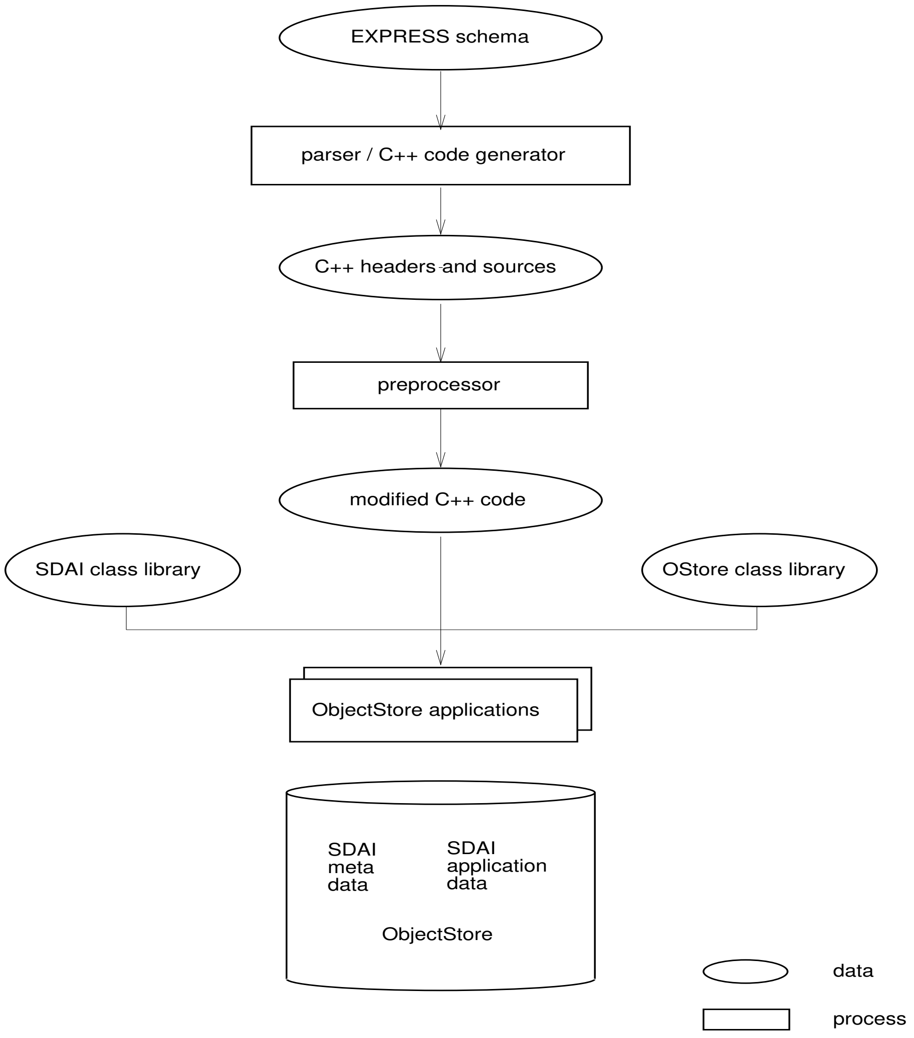

Figure 1: EXPRESS/SDAI database components

2.1.1 EXPRESS parser

From [Wil93], we elected the publicly available parser developed by the National Institute of Standards and Technology (NIST at Gaithersburg, MD). This parser is coupled with a C++ code generator called *fedex_plus*. We are briefly summarizing its functionality:

- Every EXPRESS entity contained in an application schema is mapped to a C++ class. A subtype becomes a derived class. There is no support for the sophisticated inheritance specifications using ANDOR etc. but this has not been our focus anyway.
- EXPRESS explicit attributes yield C++ protected data members. Furthermore, both a put and a get method are generated for every attribute – with the same name as the attribute, distinguished by parameter overloading.
- Schema information (meta data) is preserved in special C++ code (descriptors managed by a class called Registry).

2.1.2 Preprocessor

We developed a preprocessor to transform some of the generated C++ code so that the fedex_plus output better conforms to the SDAI specification. A more tightly coupled solution, i.e. a parser–driven generation of SDAI–compliant code, would have avoided any information loss. However, it would have required changes to fedex_plus – an option we considered not worth the effort.

Our preprocessor is actually a collection of *awk* programs. One of its functions is the adjustment of the fedex_plus output to SDAI naming conventions. At the same time, the preprocessor modifies source code to take advantage of C++ templates. These substitutions are efficiently processed by *awk* as long as regular pattern matching is applicable. For example, if the type of an entity attribute in an EXPRESS application schema reads `ARRAY OF INTEGER,` fedex_plus generates `IntAggregate,` which is preprocessed to `SdaiAggr<SdaiInteger*>*.`

Other changes which are easy to handle include that the declaration of derived classes has to be *virtual* (for multiple inheritance) and that entity instances have to be created *persistently*, i.e. in the database and not just in main memory.

It is more difficult, though possible, to generate additional source code to allow for attribute–independent *put* and *get* methods. Together with a header file to support an entity–independent *CreateInstance*, the preprocessor facilitates an SDAI **late binding**.

2.1.3 SDAI class library

These general C++ classes do not pertain to a given EXPRESS application schema. Furthermore, they do not depend on fedex_plus either. Consequently, we can exchange the EXPRESS parser at a later point in time if necessary.

The main purpose of this library is to put the EXPRESS definitions of both the SDAI dictionary and session schema into C++ classes according to the N350 binding specification. Figure 2 shows parts of our class definition for an SDAI session, which takes advantage of ObjectStore collections:

```
class SdaiSession : public SdaiSessionInstance {
  friend SdaiRepo;
protected:
  SdaiImplementation* _sdai_implementation;
  SdaiSessionLog*     _sdai_log;
  os_List<SdaiRepo*>* _known_servers;  /* available SDAI repositories */
  os_Set<SdaiRepo*>   _active_servers; /* open SDAI repositories */
  os_Set<SdaiModel*>  _active_models;  /* open SDAI models */
  SdaiRepo*           _dictionary_rep; /* SDAI data dictionary */
private:
  void AddActiveServer(SdaiRepoH&);
  void RemoveActiveServer(const SdaiRepoH&);
  void AddActiveModel(SdaiModelH&);
// ...
public:
  const SdaiTransactionRepoH OpenTransactionRepo(const SdaiString& name);
//...
}
```

Figure 2: Fragments of the SdaiSession class

2.1.4 ObjectStore

We decided to use this ODBS because we found the following technical features well-suited for the implementation of the C++ binding:

- Seamless integration with C++

  ObjectStore belongs to the category of "persistent programming languages". These ODBSs avoid an language impedance mismatch.

- Persistence orthogonal to type system

  There is no special class from which you need to derive those classes that shall have persistent objects.

- Inverse member declaration

  This kind of support for referential integrity facilitates the implementation of EXPRESS inverse attributes.

- Transaction management

  This functionality is beyond N350. We expected to see some sort of database transactions, which were eventually integrated into the latest specification.

- Queries on collections

  This feature, too, is a useful extention of N350. However, SDAI application programs can choose not to take advantage of ObjectStore queries in order to be portable. The development of an SDAI query language has been begun in the STEP Working Group 7 but it is not at all completed yet.

- Versioning

  Again, we are interested in how to exploit advanced system concepts in our prototype. In this case, we make use of ObjectStore workspaces and configurations to freeze the versions of data together with metadata (see section 3.3.1).

2.2 Binding specifics

The C++ binding captures plenty of the EXPRESS semantics [RM93]. For example, the simple form of EXPRESS inheritance maps smoothly to the C++ data model. Furthermore, operations on strongly-typed aggregates can be type-checked at compile time. The compiler can also be enlisted to check (sub)object references.

In [LH94], the authors distinguish between different styles of access and binding. It should be clear now that our implementation represents a **direct binding** rather than an *upload/download* or a *cached* one. It is due to the seamless integration of an SDAI application with both the SDAI and ObjectStore class library that an individual SDAI access does usually not result in a "high-cost database operation" as generalized in [LH94]. The Virtual Memory Mapping Architecture of this ODBS [LLOW91] allows object traversal at a speed of main memory access once the page containing a persistent object is mapped into the client cache [LLOW91]. Strictly speaking, our EXPRESS/SDAI database *is* a cached implementation: physical parts of a database (segments, pages) storing entity instances are cached transparently by ObjectStore.

The major source of performance loss is the SDAI metadata management. Populating and maintaining the SDAI dictionary cannot be waived if the SDAI application insists on an SDAI late binding. By a symbolic constant (compiler switch), we can turn off the late binding option. This implies, however, that we have to give up the flexibility to write schema-independent applications. Moreover, the SDAI metadata support is a prerequisite for future data interpretation. Our further studies on SDAI–integrated archiving will make use of our prototype's metadata management.

3 SDAI–integrated archiving

Before we describe our approach of how to archive data which is stored in an EXPRESS/SDAI database, we look at general reasons for archiving. Our concept of SDAI–integrated archiving should be contrasted with traditional archiving techniques for both files and databases. We consider *application–oriented database archiving* to be a special long–term database support [Her94].

3.1 Traditional archiving

Depending on the domain in which people work, there are basically three answers to the question "Why do you archive something?":

1. To keep information over long periods of time

   More specifically, long-term archiving can either be required by law e.g. to prove the correctness of a product's design or may have an economical motivation e.g. the reuse of engineering know–how [HM94].

2. To not waste expensive storage for less valuable data

   From a technical standpoint, this data is less frequently accessed or can afford longer access times. Storage (media) economy is addressed here [ZPD90].

3. To have a backup copy for recovery

   This is a typical database issue. Database systems usually have an archive utility, which dumps (copies) pages to some backup medium. The data to be archived remains in its original place [MN93].

The *objects subject to archiving* are paper documents, electronic data, and finally data which is managed by a database system. However, a fundamental difference between archiving files (answers 1 and 2) and databases (answer 3) appears: on the one hand, the archived data is logically separated from the "active" data by a user. On the other hand, there is redundant data, which is copied in system-internal units (pages, "dbspaces", or entire databases) without any knowledge of application objects.

3.2 Application-oriented database archiving

The previous observation leads to the following challenge to database systems: make archiving an integrated service which allows an application program to **deliberately** indicate by means of the **data model**, which data

- is going to be accessed rarely and can thus be put on slower but inexpensive storage
- should be hidden until access to it is explicitly required again!

For simplicity, we assume that both characterizations specify the same data set so that deferring the actual data transfer is an implementation choice. The main point is that we propose a "DML like" operation which can be invoked by a user or administrator and which archives data in the granularity according to the DDL.

To illustrate the latter, we refer to a suggestion made by Codd for the relational data model (RM/V2). In [Cod90], he calls for an ARCHIVE operation which puts data (tuples) from SQL tables (complete relations or views) to long-term storage. Before this data can be accessed again, the database administrator must have issued a RESTORE command. If a table with same name exists in the current database, it is overwritten when reloaded. We consider this approach a first start but it falls short of usual backup/restore.

3.3 Archiving in terms of EXPRESS and SDAI

3.3.1 New SDAI operations

Applied to an EXPRESS/SDAI database, application-oriented archiving means extending the set of SDAI operations. We first introduce two new operations and then deal with the access to archived data.

1. SELECT

   This operation identifies the data to be archived. The selection is done in units defined in the EXPRESS language. An SDAI–integrated implementation can take advantage of the SDAI metadata management. In fact, pointers to the following collections of data can be used as parameters of the SELECT operation:

   - With respect to the SDAI Dictionary Schema

     A schema and a (possibly complex) entity are potential EXPRESS units of archiving. In detail, selecting a set of entities cannot be done arbitrarily. For example, global rules might cover certain entities. Non-inheritance relationships could also stipulate that some entities be archived together.

   - With respect to the SDAI Session Schema

     Instances making up an entity extent, an SDAI Model, an SDAI Schema Instance, or an SDAI Repository could be destined for archiving. We believe that SDAI users will group semantically connected data in a Schema Instance, which is therefore our favorite granularity. Again, a sequence of SELECT operations may be required to compromise between the user's specification and the system's check on the validity of the data set. Open references between selected and unmarked data could also prevent an initial choice from becoming effective.

2. ARCHIVE

   This operation archives data which has been selected successfully . To store this data at a reasonable cost, it is moved from disk to tertiary storage. Examples include optical disk jukeboxes or tape libraries [Wal94, KPCD+92]. At the same time, data can be compressed and can change representation to adjust to the new device characteristics [Sto91]. As mentioned above, it is advisable not to migrate the data immediately but at a convenient point of time, e.g. when the work load is low. Logically, we assume a synchronous semantics of the ARCHIVE operation.

Contrary to [EL90], we do not advocate a uniform view of active and archived data in terms of **visibility**. In the engineering database prototype PRODAT, one has navigational access to database objects. Data referenced at run-time is transparently restored from an archive, which needs to be on-line. We, however, focus on applications in which the archived data is excluded from the current work database (e.g. a released product design).

For the interpretation of the data later on, it is essential to archive the metadata as well. The structure of an EXPRESS application schema, for example, is represented by instances of the SDAI dictionary schema. If the application schema evolves over time, the original version of the schema can at least be frozen in the archive. In contrast to **moving data**, we argue to **copy metadata** during archiving. This is because metadata is less bulky and likely to be reused immediately.

The conceptual separation of archived data from active data does not prevent the SDAI user from applying the **same access operations** including queries to either kind of data. Access time, of course, will differ. There is no need for an SDAI RESTORE operation as we will see in section 3.3.2. The redirecting of data from tertiary to secondary storage is only recommended when extensive reuse is going to happen or updates-in-place are requested but not supported by the archive device. Actually, archive updates are controversially discussed. On the one hand, updates contradict most people's philosophy of an archive. On the other hand, even conventionally archived technical drawings undergo corrections provided that the changes are documented. In this case, designers were afraid of loosing functionality.

3.3.2 SDAI archive repositories

Besides new operations, we suggest the following extension of SDAI. It is a straightforward approach to establish a special repository which serves (logically) as archive and which resides (physically) on tertiary storage. The SDAI session schema would look like this:

```
ENTITY sdai_session;
 known_servers:   SET OF sdai_repository;     -- usual repositories
 archive_servers: SET OF archive_repository;  -- archive repositories
 active_servers:  SET OF sdai_repository;     -- opened reps (for both)
  ...
END_ENTITY;
```

The concept of an *archive repository* fits the specified data separation. With respect to figure 2, we need to implement `os_Set<SdaiRepo*> _archive_servers`. Opening and accessing an archive repository only requires operations the SDAI user is familiar with.

4 Conclusion and future work

The SDAI–integrated archiving of data as specified in this paper is a long-term service of growing importance. Its implementation within a database system is a technological challenge. A major problem is the full support of tertiary storage by a database system [CHL93, Ols92]. Once tertiary storage can be accessed the same way as secondary one (neglecting performance aspects), it can host SDAI archive repositories. As long as the extensibility of database systems at the storage layer is limited, we need to simulate the corresponding storage operations. We are currently investigating this kind of openness in other ODBSs. Aside from this, our future research includes the study of implementation options for different kinds of SDAI archive repositories such as read-only archives for legal (e.g. liability) purposes.

References

[CHL93] M. J. Carey, L. M. Haas, and M. Livny. Tapes hold data, too: Challenges of tuples on tertiary store. In *ACM SIGMOD*, pages 413–417, Washington, 1993.

[Cod90] E. F. Codd. *The Relational Model for Database Management: Version 2.* Addison–Wesley, Massachusetts, 1990.

[DHSV94] M. Dach, N. Hoimyr, J. Saarela, and J. Vuoskoski. Using EXPRESS in a high energy physics research environment. In *4th Int. EXPRESS Users Group Conf.*, Greenville, October 1994.

[EL90] J. Encarnacao and P. C. Lockemann. *Engineering Databases.* Springer, Berlin Heidelberg New York, 1990.

[Her94] A. Herbst. Long-term database support for EXPRESS data. In *7th Int. Working Conference on Scientific and Statistical Database Management*, Charlottesville, VA, September 1994.

[HM94] A. Herbst and B. Malle. Core functions of a long–term archiving system for product data. In *27th International Symposium on Automotive Technology and Automation, Dedicated Conf. on Mechatronics*, pages 381–388, Aachen, November 1994.

[ISO94] ISO. *10303-22: Product Data Representation and Exchange - Part 22: Standard Data Access Interface.* 1994. N350.

[KPCD+92] R. H. Katz, D. A. Patterson, A. Chervenak-Drapeau, J. Fine, and E. Miller. An approach to cost-effective terabyte memory systems. In *COMPCON Spring'92*, San Francisco, February 1992. IEEE Computer Society Press.

[LH94] D. Loffredo and M. Hardwick. Efficient database implementation of EXPRESS information models. In *4th Int. EXPRESS Users Group Conf.*, Greenville, 1994.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The Objectstore database system. *Communications of the ACM*, 34(10):1–14, October 1991.

[MN93] C. Mohan and I. Narang. An efficient and flexible method for archiving a data base. In *ACM SIGMOD*, Washington, 1993.

[Ols92] M. A. Olson. Extending the Postgres database system to manage tertiary storage. Master's thesis, Univ. of California, Berkeley, 1992.

[Owe93] J. Owen. *STEP - An Introduction.* Information Geometers, Winchester, 1993.

[Pri93] A. Primbs. STEP/EXPRESS data management using an object-oriented database system. Diploma thesis, IBM Heidelberg Scientific Center/Univ. of Mannheim, December 1993. in German.

[RM93] T. Rando and L. McCabe. Object–oriented approaches to the implementation of EXPRESS–based systems. In *3rd Int. EXPRESS Users Group Conf.*, Berlin, 1993.

[SB93] F. Schoenefeld and C. Bohm. Using EXPRESS database technology for accessing NCBI genomic data. In *3rd Int. EXPRESS Users Group Conf.*, Berlin, 1993.

[Sto91] M. Stonebraker. Managing persistent objects in a multi-level store. In *ACM SIGMOD*, pages 2–11, Denver, 1991.

[SW94] D. A. Schenck and P. R. Wilson. *Information Modeling: The EXPRESS Way.* Oxford University Press, 1994.

[Wal94] S. Wallace. Managing mass storage. *Byte*, 19(3):78–89, 1994.

[Wil93] P. R. Wilson. EXPRESS tools and services. Rensselaer Polytechnic Institute, Troy, December 1993.

[ZPD90] P. Zabback, J. B. Paul, and U. Deppisch. Office documents on a database kernel – filing, retrieval, and archiving. In *5th Conf. on Office Information Systems*, pages 261–270, Cambridge, April 1990.