IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 January 2026, accepted 9 February 2026, date of publication 16 February 2026, date of current version 19 February 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3664594

RESEARCH ARTICLE

Preserving Long-Term Access to Decommissioned Database Systems With Immortal Database Access (iDA)

ELADIO GUTIÉRREZ[1], IVAR RUMMELHOFF[2], SERGIO ROMERO[1], THOR KRISTOFFERSEN[2], JOSÉ A. TIRADO-DOMÍNGUEZ[1,3], MARIA DEL CARMEN LÓPEZ[3], AND OSCAR PLATA[1]

[1]University of 29071 Málaga, Málaga, Spain
[2]Norwegian Computing Center, 0314 Oslo, Norway
[3]Tedial S.L., 29590 Málaga, Spain

Corresponding author: Eladio Gutiérrez (eladio@uma.es)

This work was supported in part by the Eurostars-3 through the Immortal Database Access (iDA)–Long-Term Recovery and Access to Decommissioned Database Systems, granted by European Union under Project E!1622; and in part by the Open Access funding provided by Consorcio de Bibliotecas Universitarias de Andalucía (CBUA) and University of Málaga.

ABSTRACT When a database system is decommissioned, retaining its data, structure, and query capabilities is often crucial for future restoration and access. The Immortal Database Access (iDA) solution focuses on preserving decommissioned databases along with their stored information and retrieval functionalities. Building upon the Immortal Virtual Machine (iVM) technology, iDA provides tools that ensure long-term preservation of databases on physical storage media. This includes not only safeguarding the stored content but also enabling its regeneration with functional search capabilities. For this purpose, two innovative elements are introduced: *DbSpec*, a new language for managing the decommissioning process, and a *Read-Only Access Engine (ROAE)* which serves as an interface to future users who wish to retrieve the decommissioned information stored on the long-term substrate. ROAE complements the SIARD (Software Independent Archiving of Relational Databases) standard. Although SIARD is effective at preserving database data and metadata, it lacks the ability to capture the essential search and query functions necessary for meaningful information retrieval. iDA addresses this limitation, ensuring that decommissioned systems remain accessible and functional for future users.

INDEX TERMS Database systems, digital preservation, formal specifications, information representation, application virtualization.

I. INTRODUCTION

The Immortal Database Access project (iDA)[1] explores the long-term preservation of relational database systems in immutable physical storage, particularly after these systems have been decommissioned.

A. MOTIVATIONS

Relational databases have been the dominant solution for storing and managing interconnected and related data for decades, having been widely adopted across industries. In both the public and private sectors, databases and database-driven systems must be archived for extended periods, or in some cases indefinitely, due to legal or business requirements. This archiving can be done periodically or at the end of their useful life. Keeping data accessible may be key to regulatory compliance. Generally, when archived with current tools, only the data is captured, so users may lose the query functionality and context in which it is used.

The associate editor coordinating the review of this manuscript and approving it for publication was Genoveffa Tortora.

[1]EU Eurostars project "iDA: Immortal Database Access – Long-Term Recovery and Access to Decommissioned Database Systems". All software is publicly available in the repositories listed in appendix.

This issue is especially critical if the archiving period is very long, as the original software or the people who knew how the system operated may no longer be available. In addition, events such as technical obsolescence, migrations, hacking, or loss of content may occur during the archiving period, resulting in the loss of the practical value of the preserved data to the users.

B. HOW DOES THE iDA SOLUTION HELP?

iDA allows users to decommission and preserve database systems in a flexible way, independently of the software that originally created or managed the data. This involves preserving not only the data itself, but also the ability to query it and extract results, providing future users with the context of how the content is accessed.

Use cases for our solution include accessing records about individuals, such as their educational and health information, or cultural heritage information, or research data, etc. Another important example would be accounting and business information created in enterprise management systems such as SAP. In many cases, access to this data will only be necessary infrequently or when requested by authorities, government agencies or during legal disputes. Therefore, it is desirable to offer a cost-effective and secure solution that avoids any future risk of constant data migrations to keep the data alive in the long term.

C. WHAT DOES THE iDA SOLUTION PROVIDE?

Most long-running systems use some form of relational database to manage their data. An industry standard for archiving such systems is the SIARD [1] format. Our approach has been to build a solution on top of this specification, but with a number of important additional features that are not part of this storage standard. The solution has two main components: an executable specification language, DbSpec, and an interface to the preserved data, ROAE, which in turn builds on the technology iVM [2] for preventing digital obsolescence.

A DbSpec specification is a precise description of the steps needed to preserve a database or other relational data. The result is usually an information package containing one or more SIARD files, a file describing expected future use cases to the ROAE interface, the specification itself and additional metadata. When possible, the package should also contain the original database backups and other input data. Thus, we not only document how the archive was compiled, it can also be reproduced by (re)executing the specification. This is especially valuable when data from other sources or complex data transformations are needed.

The *Read-Only Access Engine* (ROAE) allows future users to interact with and retrieve both functionality and data from decommissioned databases. ROAE operates on SIARD files following the use cases generated by the archivists through DbSpec. This requires a new file format, to store the use case information, which must also be preserved. This information includes: a description for the user of each use case itself; possible variable parameters to be defined by the user when running such a query; and a formal expression of the query to be handled by the chosen database engine.

Finally, to ensure the very long term survival of the data, the iDA solution uses the storage medium on photographic film *piqlFilm*, together with the support of the *Immortal Virtual Machine* (iVM) developed in a previous project [2].

II. BACKGROUND

A. THE iVM ECOSYSTEM

The *Immortal Virtual Machine* (iVM) ecosystem was introduced in the context of an information preservation solution aimed at guaranteeing the access to today's digital data in the future (for hundreds of years).

In particular, this solution stores the data on a preservation medium using technology developed by the company Piql AS, where the information is digitally encoded in frames on a high-resolution photographic film. Figs. 1–2 show the information preservation flow and outline the layout of the film frames respectively.

The role of the abstract virtual machine is to implement the format decoders, which is stored in binary along with the data to be preserved. This results in self-executing content that can produce consistent results (images, sounds, etc.) on the output devices. One major design requirement of the iVM architecture was to have a simple formal description that would allow future developers to easily implement it.

The iVM-based preservation solution includes three main components:

1) the abstract machine extremely simple in order to be formally described (iVM),
2) an independent and technologically neutral descriptions of the machine, preserved in analog form and intended for future developers,
3) a complete toolchain.[2]

The toolchain is responsible for generating iVM binary code from the source code of the decoders, assuming that these decoders are written in C or C++. An important component of this toolchain is the compiler. From the many available C/C++ compiler infrastructures, GCC (GNU Compiler Collection) [3] was chosen [2] due to its popularity in the community and a fairly stable development API. The iVM toolchain is completed with the standard C library (the *newlib* library [4], [5] was ported) and a software-based floating-point library.

B. THE iVM ARCHITECTURE

The Immortal Virtual Machine is an abstract 64-bit processor (referred to as the *iVM* architecture) designed with the goal of being easily described in a precise way so that future developers can recreate it and run the software necessary to extract the information preserved on the film reel along with

[2]GCC compiler for C/C++, assembler and emulator targeted at this abstract machine.

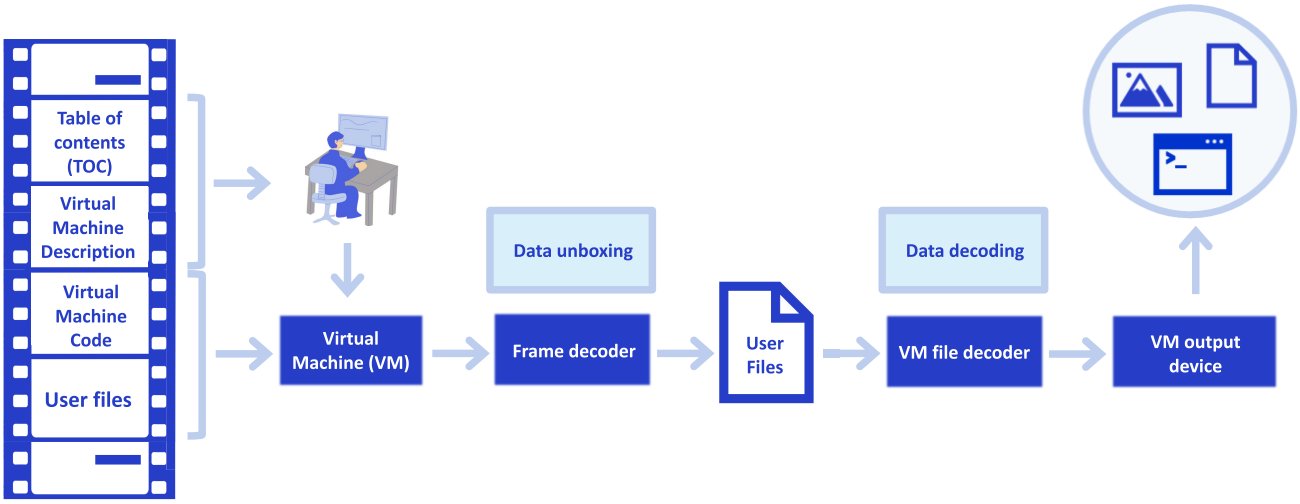

FIGURE 1. Flow of the codification and decodification of information in the film.

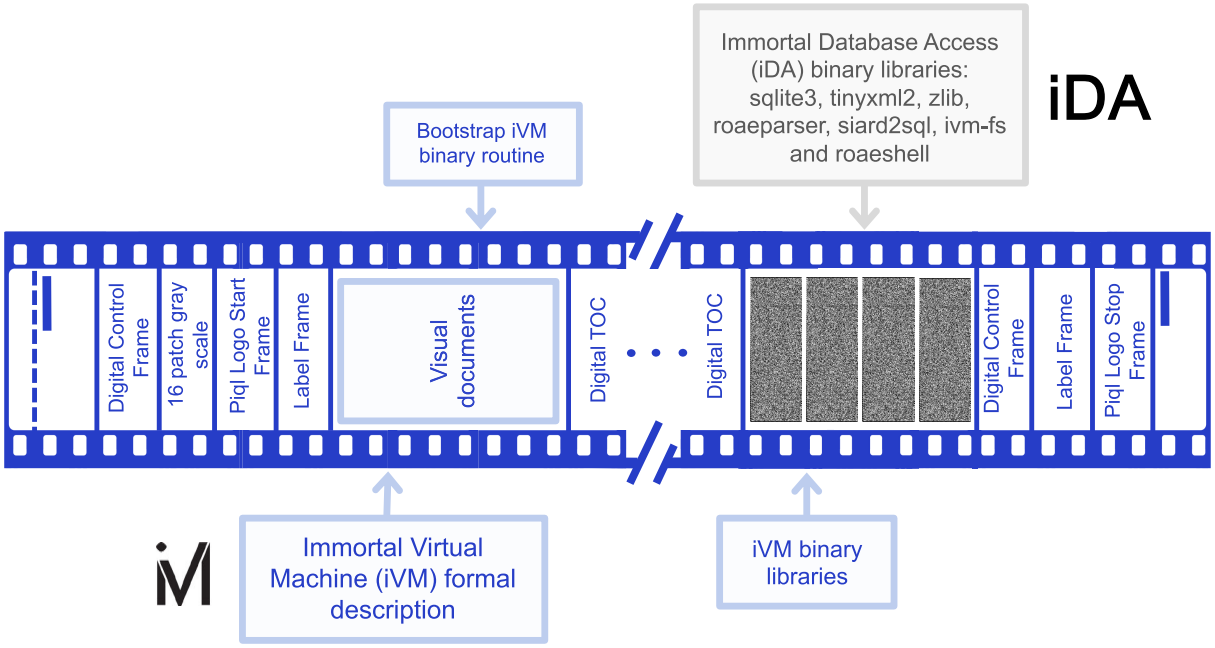

FIGURE 2. Format of data in the film.

the software itself. It is a 64-bit stack machine with a minimal design consisting only of these elements:

- a byte-addressable memory of arbitrary size
- two registers: the program counter (PC) and the stack pointer (SP)
- a stack-based reduced instruction set
- a simplified I/O system.

Immediately after powering the machine on, the PC register points to the first memory location (address 0), SP points to the last memory location, and the stop bit is reset to 0. On this machine: the stack grows to decreasing addresses, registers are 64 bits wide, the memory is byte-addressable, and data is stored using an *little-endian* scheme. Execution is terminated with an exit instruction, which sets the halt bit.

The iVM input/output system uses specific machine instructions and was originally intended to render and play the multimedia content stored on the film. The output devices include a video output (screen), an audio output, and a Unicode UTF-32 text output (text console). The only input device is an image stream input corresponding to the frames read from the film itself.

C. DECOMMISSIONING PRESERVATION

Getting rid of an information system is usually easy. With modern cloud solutions a few clicks in a web interface might

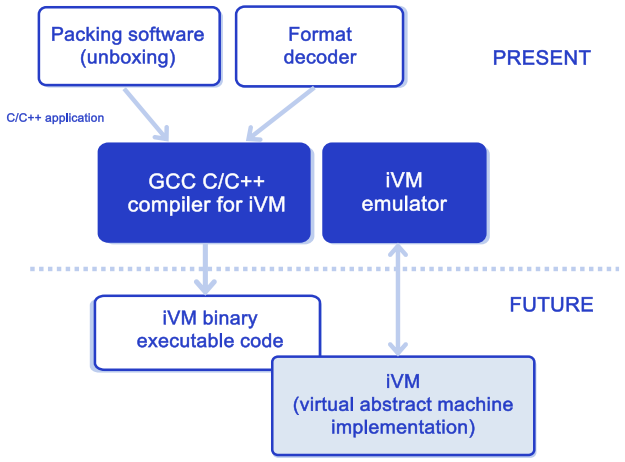

FIGURE 3. Role of the iVM compiler in the generation of the contents to be preserved.

```
Command:
    title = "Get all users"
    Parameters:
    Body:
        SELECT * FROM user_table

Command:
    title = "Get users data by ID"
    Parameters:
        userid - the user identifier
    Body:
        SELECT * FROM user_table
        WHERE id=$${userid}

Command:
    title = "Count number of users by last name"
    Parameters:
        ln - user's last name
    Body:
        SELECT count(*) FROM user_table
        WHERE lastname=$${ln}
```

FIGURE 4. ROAE file with three ROAE block declarations.

be enough. With legacy, on-site solutions there is also some computer hardware that must be safely disposed of, but there is no need for complex decommissioning processes as in, say, the oil and gas industry. However, the information could still hold some value; or the organization might be required by law to keep the information for at least a certain period. By the term ''decommissioning preservation'' we mean a process for preserving data and functionality when systems such as databases are decommissioned. It can be viewed as a sub-process of decommissioning, which can be viewed as a process of transitioning from one system to another. However, it focuses on the preservation sub-process.

The decommissioning of a computer system can have a variety of motivations, from the complete elimination of the system and its functionality, to its replacement with a new system that provides renewed functionality to users. Decommissioning is an orderly process, with a clearly defined beginning and end, and is the result of a conscious decision by someone with the appropriate authority, usually the owner of the system. A common problem is posed by software that is abandoned and/or not properly maintained over time. This poses a double risk: maintaining software that is no longer needed is a cost overrun, and running unmaintained software is dangerous.

Over time, security vulnerabilities may not be updated, and when a component necessary for some tasks stops working, it may be difficult to find a person with enough expertise to fix the problem. Therefore, it is necessary to maintain a clear distinction between the software of the system in production and that of the systems to be decommissioned, since in the latter it is necessary to ensure its future use, regardless of the software evolution. This is the main reason for the decommissioning preservation process.

D. SIARD

SIARD (*Software Independent Archiving of Relational Databases*) is an open format developed by the Swiss Federal Archives for archiving SQL-compliant relational databases, including metadata. The SIARD format was created to archive relational databases in a platform-independent manner. A SIARD archive is essentially a ZIP container of XML files. One of these XML files contains the database structure, and for each table there is an XML with its contents. We can also find text and binary files for CLOBs (*Character Large OBject*) and BLOBs (*Binary Large OBject*), respectively.

III. OVERVIEW OF THE iDA SOLUTION

The iDA solution is based on three pillars: DbSpec, SIARD, and ROAE. DbSpec is a language specifically for extracting, transforming and packaging data for preservation, as well as documenting the process. Its main usage scenario is decommissioning, but it can also be useful in other data preservation situations. In fact, since DbSpec scripts are executable, they are especially useful when the preservation task must be repeated at regular intervals.

DbSpec primarily describes the steps necessary to produce the results of data preservation. In many cases, it is not simply a matter of preserving the database(s) of the decommissioned system as is, but a curated database – possibly based on multiple data sources – that needs to be independent of the decommissioned software.

The DbSpec result consists of two distinct outputs: on the one hand, SIARD files with the contents of the relational database, and on the other hand, ROAE files, with information about expected future use cases. The DbSpec language and interpreter fill the gap between existing tools for creating and processing SIARD files by making the steps for creating the file explicit and repeatable. These steps can include packaging the resulting SIARD files and other additional files as an information package (OAIS).

Since SIARD file is not an operational database, but a static snapshot of the database contents, the decommissioning

process should also preserve information about the expected future use cases. For this we use ROAE files. These are also generated when executing DbSpec specifications; and each ROAE file is essentially a collection of use cases, called *ROAE blocks*. Fig. 4 illustrates the syntax of these blocks with an example. The block starts with a title of the use case, followed by a list of parameters (which is optional). The core of the block is a parameterized SQL query describing the use case.

This is where the iVM comes into play in the preservation solution: A *Read-Only Access Engine* (ROAE) has been developed for the iVM machine. Once this engine is included in the film, it can be used to extract information from the SIARD contents according to the use cases described in the ROAE blocks. In this way the use cases are preserved in an operational form for future use.

We could say that the ROAE engine extends the multimedia renderers originally included in the iVM project by providing support for designing user interfaces to access the preserved databases. Note that such databases may contain multimedia information, such as PDFs, images, videos, or others, in which case the renderers already developed for the iVM architecture are used to render them.

IV. DBSPEC

DbSpec consists of (1) a domain-specific language for specifying database preservation and (2) a tool for executing such specifications. Even though DbSpec was developed as a part of iDA, it can also be used for database preservation which does not involve ROAE or iVM. This has been described in some detail in a separate article [6]. The tool itself is available as free software at the URL included in the Appendix. At that URL you will also find a complete language reference.

A. OVERVIEW

For decades, relational databases have been the core technology for storing and organizing data; but most databases are deeply integrated with the software systems used for interacting with the data – often to the extent that the database is impossible to understand without these system or their program code. In general, preserving the database ''as is'' is not enough to preserve the information. Also, the database may contain data one is not allowed to preserve, or large amounts of data with little or no value. These are all arguments for transforming and adding metadata to the database during the preservation process, as long as the process is well-documented and reproducible. This is where DbSpec comes in. Here are some tasks one can use DbSpec specifications for:

(a) Attach documentation and other metadata to columns, tables, views and the database as a whole.
(b) Rename database objects with misleading or unintelligible names.
(c) Remove information that should not be preserved, including obsolete or temporary tables.
(d) Verify assumptions regarding the data, e.g. that some combinations do not occur.
(e) Formalize representation invariants in terms of database constraints.[3]
(f) Import data needed to understand this database from other sources, such as files, services or other databases.[4]
(g) Replace or remove uses of obsolete or non-standard SQL features.[5]
(h) Normalize the database in order to save space and simplify representation invariants.
(i) Improve the data quality, e.g. by fixing inconsistencies or removing duplicate data entries.

B. EXECUTABLE SPECIFICATIONS

DbSpec is a language for writing executable specifications of database preservation steps. More precisely, a specification is a text file which will typically have the following structure:

1) a list of parameter declarations;
2) en embedded script for restoring a ''native'' database backup to a *staging database*;[6]
3) a statement establishing a connection to this database;
4) assertions to ensure that our initial assumptions about the database are correct;
5) embedded scripts for importing additional data;
6) embedded scripts for transforming the staging database;
7) assertions to ensure that the staging database now has the properties we want;[7]
8) ROAE blocks (cf. Sec. III);
9) statements for specifying or adjusting SIARD metadata;
10) one statement for generating (i) a SIARD archive of the staging database and (ii) the corresponding ROAE file;
11) an embedded script for compiling an information package containing the SIARD archive, the.roae file, and other relevant information – perhaps even the initial database backup.

In order to execute the specification, we use the corresponding DbSpec interpreter,[8] but observe that many of the steps involve other tools via *embedded scripts*. These come in two flavors: SQL scripts must be executed on a database server via a database connection, whereas non-SQL scripts will use the current execution context and languages such as Python, Perl, Bash or Powershell. The languages used will depend on the context. Fig. 5 shows a small

[3]For example, many legacy databases have no foreign key constraints so that the system can perform insertions and deletions in any order.
[4]A typical example would be to create a table that maps status codes to textual descriptions.
[5]Otherwise, information may get lost when creating a SIARD archive of the database.
[6]In general, native backups use proprietary formats of the database vendors. Unlike portable formats, these can ensure that we get an exact copy of the original database.
[7]By restoring the initial database backup to a second database, we can also write assertions relating the staging database to the contents of the original database backup.
[8]On Unix-like systems one can also make the file itself executable by including an interpreter directive (#!) as the first line.

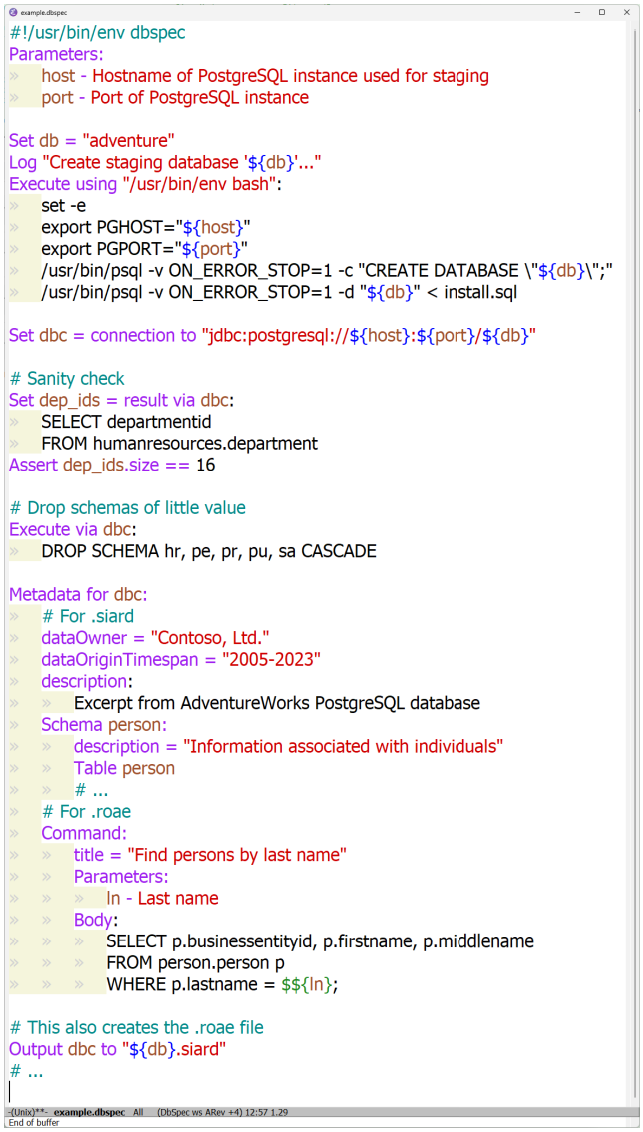

FIGURE 5. Example DbSpec specification open in Emacs.

DbSpec specification being edited in the Emacs text editor). In real-world examples, the specifications will be quite a bit longer.

C. EXECUTION ENVIRONMENT

In general, a DbSpec specification should be executed in an environment which is as similar to the production environment as possible. We shall refer to this as the *staging environment*. Most importantly, the database server used for the staging database should have the same software, version numbers and settings as the production server. Otherwise, data might be lost or corrupted. Also, any additional data imported from files, other databases or the APIs of other systems must be the same as in the production environment. For example, if the system we want to preserve relies on a service for mapping addresses to physical locations and we want include these in the preserved database, then the staging environment should include a service which responds with the exact same locations.

In some cases it is also important to use the same operating system, language settings, versions of installed utilities, etc. Furthermore, the most efficient way to interact with the staging database and other data sources initially may be through libraries and tools that are part of the system to be decommissioned, in which case the staging environment must include much of this software as well – possibly even running as services. In fact, the simplest solution might be to execute the DbSpec script in the production environment itself if this can be done in a controlled and safe way.

D. DEVELOPMENT PROCESS

We recommend developing DbSpec specifications through multiple iterations as a subprocess of the decommissioning process mentioned in Sec. II-C. For legacy systems, making sense of the data can require a lot of effort. Thus, we recommend following an iterative plan-do-check-adjust process. When using DbSpec, parts of this process can be automated. Each time a draft specification is executed, the result set can be checked against the decommissioning requirements, and any shortcomings can be adjusted before the specification is re-executed. This is repeated until the decommissioning requirements are met. Thus, the final DbSpec specification will provide a formal documentation of all the steps of the decommissioning process at the technical level.

V. FUTURE DATA ACCESS

A. INITIAL CONSIDERATIONS

After the decommissioning process, two objects are generated: the SIARD file with the database contents selected to be preserved, and an associated ROAE file with the description of the use cases. The question now is how to access this information once the original system is no longer available. Several scenarios can be considered:

- An instrumental/debugging access during the decommissioning and/or preservation phase on film, to check that the process has been correct and that what has been established with DbSpec has been properly preserved.
- Access by archivists in the short to medium term; in this case, although the original systems may not be available, versions, probably more modern, of the technologies on which they were based (CPUs, file storage, operating systems, etc.) are still accessible.
- Access by archivists in the very, very long term: in this case we do not know exactly which technologies are available. Regarding the information preserved from the original database, only the SIARD and ROAE files are stored on the final film.

  Note that this medium also contains the description of the iVM machine together with the binaries for this architecture for retrieving and rendering of objects present in the preserved database. Future developers are responsible for implementing the abstract machine on which the binaries will run.

In any case, the end user is interested in interacting with the preserved content and we are presented with several alternatives for this interaction:

Textual Menu-Based Interface This solution allows users a step-by-step access to predefined queries or use-cases. Archivists can design workflows as nested menus that invoke specific ROAE query sets. It is simple and highly compatible with the ROAE goals such as being easy to describe and preserve as structured code. It maps well to preserved system functionality and typical use-cases (query selection, filtering, navigation, etc.). Additionally, it requires few resources, which is ideal for constrained environments.

The main drawback of this approach is its limited flexibility for dynamic input, for example with complex database queries.

Textual Shell / Command Line Interface (CLI) With a textual command line interface or shell, users can input arbitrary queries or commands. This makes this approach ideal for technically skilled users (developers or archivists) and also suitable for automation. This solution fits well with the concept of parametrized *ROAE* blocks, as we can invoke them as a command, providing high flexibility.

As drawbacks of this approach we can mention a steeper learning curve for non-technical users, and from the development viewpoint it requires a well-designed command language and help system.

Graphical User Interface (GUI) A GUI solution may be intuitive for most users, and may directly support multimedia data such as video, images or sound; it could even mimic the original system's GUI using the preserved assets.

Nevertheless, this approach does not fit well with the minimal boot scenarios for which iVM was designed, including its basic graphical input/output supported by iVM. In addition to being resource intensive, a GUI is harder to preserve in the long term due to its complexity, as it requires adding specific renderers (fonts, layout engines, etc.) beyond those required by the preserved information.

This approach may be interesting in short-term preservation contexts, where fully immersive reconstructions of legacy applications are needed or where the original system had a GUI and the preservation aims to accurately simulate it, especially in current technology environments.

Thus, on the one hand, the iVM goals (simple description, reproducibility, etc.) need to be paired with the ROAE concept to provide preserved functionality and data access for future users. On the other hand, short-term access may be required for preparation and development/debugging purposes, as well as medium-term use using current technologies.

As a proof of concept, a mixed CLI/menu-based solution for *rendering* the ROAE files has been implemented. The aim is to prove that an interactive application can remain accessible in the future, something that has been validated in current implementations of the iVM architecture. Furthermore, it is possible to compile and run this application on current systems, which has also been successfully tested. This way, this proposal fits into a *fly what you test, test what you fly* approach. Such a mixed solution has been called *ROAE shell* and its details are described in the following subsections.

B. THE ROAE SHELL

The *ROAE shell* has been developed as an interactive application to reproduce the use cases of a decommissioned system whose data has been preserved in SIARD format on the film support. Let us look at it in the context of the iVM machine, regardless of the fact it could be compiled and executed as a conventional application on current computer systems.

During the execution of the reel bootstrap procedure on the iVM machine, the SIARD files digitally preserved on the film are transferred to an in-memory file system and converted to a SQL format which is compatible with the selected database engine (one supported by the SQL dialect of the ROAE blocks).

The startup procedure ends with the launching of the ROAE shell, which provides the (future) user with a way to execute the use cases that – in the decommissioning procedure through DbSpec – it was decided to include as ROAE outputs. Interaction with the shell is done via two of the input/output devices defined for the iVM machine: the text console and keyboard input (see Subsection V-D).

C. GOALS AND ARCHITECTURE

The ROAE shell acts as an interactive CLI interface where requests are entered through a keyboard to obtain a response. The data source consists of SIARD files coming from a decommissioned database, and the requests correspond to ROAE use cases. Fig. 6 shows the block organization of the shell.

From the viewpoint of the ROAE shell development, several challenges needed to be addressed. First, it should be considered as a piece of software primarily for future users, whose binary will be preserved along with the data in the iVM machine format. This imposes some constraints derived from the toolchain developed for the iVM architecture. Consequently, such software must consist of a self-contained set of C/C++ sources, with no dependencies other than the standard C/C++ libraries or other libraries fully included in the toolchain.

It is also important to consider that the developed solution may be used in the present, so it should be possible to build it on current or near-future machines. As an additional requirement, the ROAE shell needs to be based on open solutions and libraries that are freely available.

A key design consideration was the choice of the database engine that the shell would use internally, since the actions

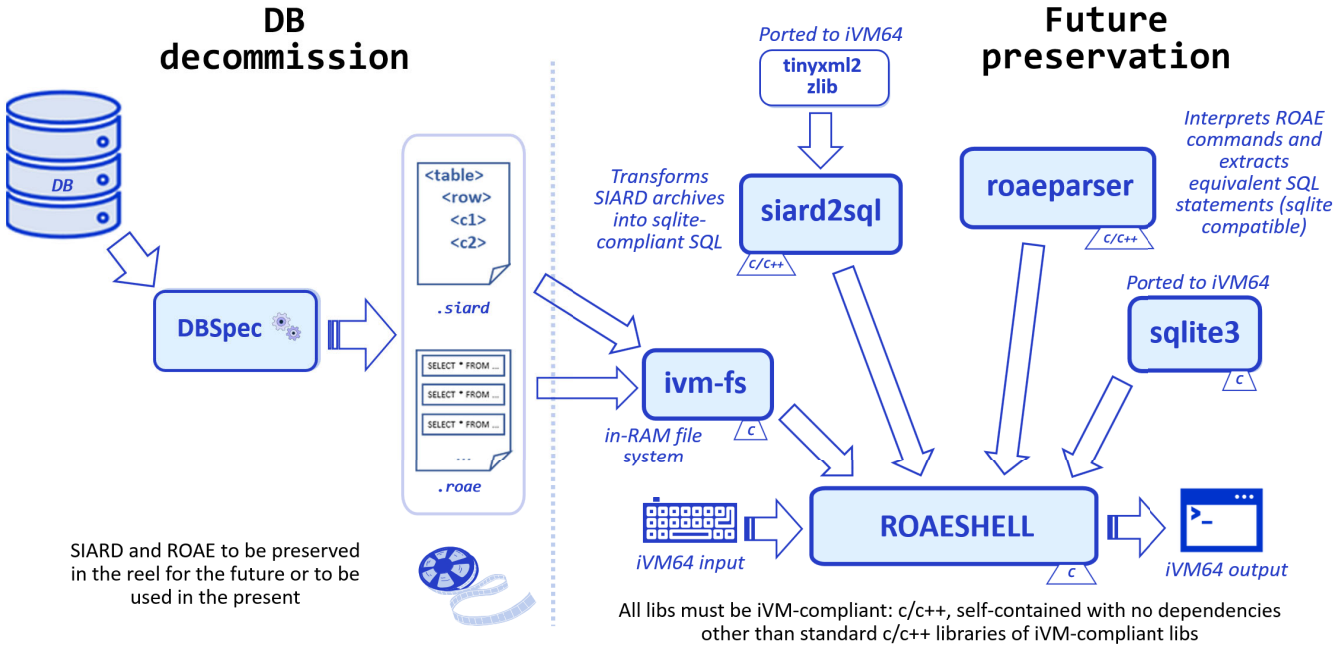

FIGURE 6. ROAE shell organization and implementation on the iVM architecture.

of the ROAE blocks must be expressed in the SQL dialect of such an engine. The chosen option was SQLite3 [7], as it solves the challenges and meets the requirements while offering several advantages:

- It is written entirely in C,
- it is a public domain software,
- it is small, fast and self-contained SQL engine,
- it is highly reliable,
- it uses a fairly complete dialect of SQL,
- it is very popular, and
- it is expected to have long-term support.

It is also worth mentioning that SQLite is one of the database formats recommended by the Library of Congress [8].

The shell has been developed in C and integrates the four key components shown in Fig. 6: the SQLite3 database engine, the IVMFS file system, the *siard2sql* conversion library, and the ROAE parser.

A SIARD to SQLite-compatible SQL conversion library was necessary due to two major issues with the existing SIARD tools [9]. The first is that these tools are written in Java with multiple package dependencies and thus exceed the limitations of the iVM toolchain. Secondly, although these tools have support for a wide range of database engines, a specific SIARD to SQLite conversion module is missing.

Along with the *siard2sql* library and the IVMFS file system (described in Subsection V-E), the ROAE parser is another key component developed as part of the iDA solution.

The ROAE shell uses three external libraries, all written in C/C++ and openly available: *tinyxml2* [10], *zlib* [11] and *sqlite3* [7]. The first two are used by *siard2sql*, while *sqlite3* library is a core part of the shell operation. All of these libraries are self-contained, with no dependencies on anything other than the standard C/C++ libraries. Some of them have required some tweaking when ported to the iVM architecture.

D. REQUIRED iVM EXTENSIONS

The development of the iDA solution required the extension of the original Immortal Virtual Machine (iVM) [2] with the following new features:

- A new interactive textual input device for data entry from the console.
- A new machine instruction to check the virtual machine version.

The console input device uses the same encoding as the text output device, i.e. Unicode UTF-32.

The text input and output devices are mutually independent (and formalized as separate input/output monads). Therefore, no *echo* or *line discipline* features should be expected by default in these two devices. Such features typically associated with a *tty* terminal device should be emulated by the software upper layers or applications developed for the iVM if required, for example, an interactive command line interface.

E. IVMFS: AN IN-RAM FILE SYSTEM FOR iVM

To work comfortably with the iVM ecosystem in the context of the ROAE shell, a file system for iVM had to be designed. Since the machine has no non-volatile storage, it was decided to develop an in-memory file system: the IVMFS. The IVMFS filesystem [12] is generated by a shell script that

produces a C code with the initial content of the file system, consisting of the files and directories to be included. This C code contains two key elements: (1) a static initial content of files and directories, and (2) a set of primitives written in C that provide the basic file access functionality.

The resulting file is compiled and linked with the rest of the application, with the only exception that it must appear before the standard C library in the linking order. The linker places the functions declared in the file system before any other already present by default (without file system), overriding those that come from the standard C library based on newlib [4]. Thus, a complete file system is recreated in memory, which can evolve from its initial contents.

The primitives defined in IVMFS belong to the lowest level primitives (`open, close, read, write,...`) on which the higher level ones (`fopen, fwrite, fread, fprintf, fscanf,...`) of *newlib* (Subsec. II-A) are built. IVMFS supports nested directories.

Special emphasis has been placed on making the IVMFS API and its behavior as POSIX-compliant as possible. This minimizes the need to adapt to the iVM architecture of those projects using file systems, such as the libraries on which the ROAE shell is based, including SQLite.

The implemented filesystem is based on contiguous allocation, with a space reservation large enough for the size of each file. If the reserved size is surpassed while writing or truncating the file, a reallocation operation to a sufficiently large space is performed. The directories are implemented in a virtual manner, including the path associated with each file in the filename.

IVMFS also provides support for the standard C input/output streams (`stdin, stdout, stderr`). For the standard input, a simplified line discipline (tty) is also provided through the *ioctl-tty* interface. This discipline allows the user to enable *echo* to the text output (equivalent to `stty echo`), and a simplified canonical mode that allows, for example, the deletion of characters on the line being typed (equivalent to `stty icanon`), among other things. Other advanced features supported by IVMFS include redirection of open files (`dup`), symbolic links, and integration of iVM I/O devices into a `dev` directory, similar to POSIX systems.

F. ROAE SHELL USER EXPERIENCE

The ROAE shell provides a set of basic commands that allow the interaction with a database stored in SIARD format through a set of use cases specified in ROAE format. In the context of the iVM preservation solution, the ROAE shell is intended to be launched at the end of the film bootstrap process, after the SIARD and ROAE files, encoded in the frames of the reel, have been transferred to the in-RAM file system. However, it can also be compiled and run separately as an application on today's systems.

Once launched, the first step is to load the decommissioned database (SIARD) into the SQLite engine, which involves the following sequence of actions:

1) unpack the SIARD file, since it is a ZIP64,
2) convert the unzipped XMLs, together with the CLOBs and BLOBs, into a SQLite-compatible SQL format (using the *siard2sql* library), and
3) finally, load the resulting SQL into the SQLite engine.

The interaction with the ROAE use cases is carried out by the shell command `roae`, whose form of use is shown in Fig. 7. The user, present or future, can list the set of all available ROAE actions (defined and preserved by DbSpec). Each action is associated with an identifier number. The ROAE block of a particular action can be displayed by its identifier or searched by the ROAE block name.

Two execution modes are available for the ROAE use cases. The first one is based on parameter expansion in the SQL body of the ROAE block. The other one *binds* the parameters through the so-called *prepared SQL statements* [13], which brings additional advantages like avoiding code-injection attacks.

Through the `roae` command, users can also access the ROAE use cases by means of a menu-based interface. Fig. 8 illustrates an example where a simple ROAE file containing three cases is loaded. After the selection of a given case, the user is prompted with the arguments required to complete the query, along with the execution mode (parameter expansion or prepared statements).

Moreover, and primarily intended for expert users, such as today's users who know the details of the database, the shell enables the execution of SQL queries directly on the SQLite engine, in addition to the predefined ROAE queries. This is done by using the supplied `sqlite` command, whose usage is shown in Fig. 9. Therefore, a user can type something like `sqlite ''SELECT * from users''` to execute this SQLite statement on the recently loaded SIARD. In addition to SQL statements themselves, it is also possible to explore the database structure by calling SQLite's own commands, e.g. `sqlite ''.help''`, or to load a SIARD database manually. For ease of use, some shortcuts have been predefined, which are also shown in Fig. 9.

The reader is invited to test the ROAE shell in the demo site listed in Appendix.

G. FINAL CONSIDERATIONS: CHALLENGES, LIMITATIONS AND FUTURE DIRECTIONS

The development of ROAE shell has posed several challenges. One notable challenge has been the development of a full-featured dedicated C/C++ compiler targeting the iVM architecture, which was essential for meeting low-level control and portability requirements. Additionally, the libraries involved in the ROAE shell (see Fig. 6) had to be adapted or partially reimplemented in order to operate within the constraints of the iVM architecture. Another challenge has arisen from the iVM's minimal design, particularly with respect to its input/output capabilities, which was oriented to make its formal description as simple as possible.

Although the current implementation of iDA focuses on relational databases, which provide a well-defined formal

```
ivmfs:/work> roae
Usage: roae load filename
       roae clear
       roae list
       roae show <command_number>
       roae search <regexp>
       roae run-replace <command_number> param0 param1 ...
              Replace parameters in body, then execute
              Use sqlite types for parameters, e.g.: 123, 'string', X'f09f8dba'
       roae run-bind <command_number> param0 param1 ...
              Prepare SQL statement, bind parameters, then execute
```

FIGURE 7. ROAE shell: interaction with the ROAE use cases available after loading a SIARD database in the SQLite engine.

```
ivmfs:/work> roae clear
ivmfs:/work> roae load example.roae
  Read 8 commands from ROAE file 'example.roae'

ivmfs:/work> roae menu
  Available ROAE cases:
   [0] "Get all users in the database"
   [1] "Get users by userid"
   [2] "Count the number of users by last name"
   [Q] QUIT
  Select ROAE command number:
```

FIGURE 8. ROAE shell: example of the menu-based interface for a ROAE file such as the one shown in Fig. 4.

```
ivmfs:/work> sqlite
Usage:
       sqlite "<sql statement or sqlite command>"
Available shortcuts:
       sqlite -- clear
              # equivalent to ".open :memory:"
       sqlite -- load <sql_file>
              # equivalent to ".read <sql_file>"
       sqlite -- loadsiard <siard_file> [schema_filter_regex]
              # equivalent to unzip + convert siard->sql + clear + read sql
       sqlite -- tables
              # equivalent to "ANALYZE main; select * from sqlite_stat1;"
       sqlite -- table_info <table_name>
              # equivalent to "SELECT * FROM pragma_table_info('<table_name>');"
```

FIGURE 9. ROAE shell: instructions for using the `sqlite` command.

model and remain central to long-term archival contexts, other data models, such as NoSQL databases, are outside the scope of this work. On the other hand, iVM exposes a character-based I/O instruction set, that constrains the types of interactions and limits the use of graphical interfaces. While this design choice supports long-term interpretability, it restricts usability in present-day recovery scenarios. Memory usage may represent another practical constraint, as preserved data and intermediate files may be loaded into an in-memory iVM filesystem. Although this is an issue to be solved by the future users that recreates the database, it may limit particularly the future scenarios emulated nowadays. Finally, it's worth to mention that the current implementation has prioritized correctness, reproducibility and long-term viability over performance.

Several directions for future work can be considered. Extending iDA to non-relational databases would broaden its range of use to other databases that currently cannot be preserved with this solution. Improving the iVM I/O subsystem, while maintaining its long-term interpretability guarantees, may also enhance its usability. Memory usage could be reduced by redesigning parts of the data pipeline between stages instead of writing intermediate files. Overall, these future developments aim to broaden the applicability of iDA while maintaining its central goal: enabling reproducible, long-term preservation and recovery

of databases independent of specific hardware, software platforms, or operating systems.

VI. iDA EXPERIMENTAL EVALUATION

This section presents the experimental framework used to validate the proposed approach for long-term database preservation and recovery within the iDA project. The goal of these experiments was, first, to demonstrate that the iDA solution is not limited to an abstract model, but it can be applied to realistic relational databases ranging heterogeneous schemas, constraints, and data volumes. Second, these examples illustrate how the iDA approach works in practice, beyond a proof of concept, thereby supporting its applicability to real-world digital preservation scenarios. Finally, these experiments provide useful information to assess the computational requirements needed for preserving and subsequently restoring databases, offering insights into the feasibility of the approach in long-term archival contexts.

The selected databases represent a variety of use cases, including both benchmark datasets and real-world data. This allows the applicability of the iDA approach to be assessed across databases with several characteristics. The experimental results focus on functional validation and practical feasibility rather than on absolute performance metrics.

It is important to emphasize that optimizing the performance of the implementation was not one of the iDA project's objectives, which were mainly oriented towards solving the challenges that emerge from the long-term digital preservation. Therefore, the experiments have been conducted to validate the correct implementation of the concepts with feasible performance rather than to maximize efficiency. Similarly, the experiments are not intended to evaluate or compare the performance of the underlying database engines or execution platforms. Instead, the emphasis is on demonstrating how long-term preservation and recovery can be achieved in practice with iDA and, on showing the resources required for that process.

Table 1 lists the tested databases and their features: the total number of schemas, tables, rows, and cells; the number of included large objects (LOBs); and the file sizes of the SIARD and SQLite representations. This set of databases is intended to be representative of realistic scenarios, and it includes four databases of different sizes and features:

- The *Norwegian Historical Population Register (Historisk BefolkningsRegister, HBR)* [14] is a real-world population database integrating historical censuses and parish records for Norway. It links individuals across time and sources, enabling demographic and social science research on population dynamics from the 19th century onward.
- *AdventureWorks* [15] is a Microsoft-provided sample relational database modeling a fictitious bicycle manufacturing company. It represents a realistic online transaction processing (OLTP) business environment with interconnected domains such as sales, production, human resources, and inventory, and is widely used for SQL Server benchmarking, performance evaluation, and educational purposes.
- *Employees* [16] is a large MySQL sample database designed to mimic corporate human resources data. It includes extensive historical records of employees, salaries, titles, and department assignments, making it suitable for scalability testing, performance benchmarking, and analytical query evaluation.
- *Sakila* [17] is a MySQL sample database that simulates a DVD rental store. It contains a moderately sized, well-normalized schema with customers, films, rentals, and inventory, and is commonly used to demonstrate relational modeling, joins, and transactional query workloads.

The next subsections describe the experimental evaluation of the two main stages of the iDA approach. In OAIS terminology [18], the first stage corresponds to the Preservation functionality, in charge of producing the Submission Information Packages (SIPs) that are transformed into the Archival Information Packages (AIPs), while the second stage addresses the Access functionality. The preservation stage is implemented through the DbSpec workflow, which captures database structures, data, and preservation-relevant metadata. The access stage, concerning the database recreation and use case execution, is evaluated using the ROAE shell, considering both present or near-future execution environments as well as very long-term scenarios enabled by the film-substrate iVM-based solution.

A. DBSPEC EVALUATION

During the development of the DbSpec language and interpreter, we performed tests using various freely available databases. Most importantly, we developed a complex example specification for the AdventureWorks database. Later, a limited evaluation of the DbSpec approach was performed using the example database Northwind Traders for Microsoft SQL Server, which revealed several software errors that were later fixed. Finally, we did a more realistic test using the database of the Norwegian Historical Population Register (HBR). As detailed in [6], the results were generally positive, validating the functional correctness and completeness of the extracted specifications in medium/large volume realistic databases. However, working with databases that are large and slow to process does pose some usability issues. In such cases we may have to rethink how we recommend developing the specifications, perhaps starting with a database copy containing only the schema.

Table 2 shows the execution times for the AdventureWorks and HBR specifications running under Windows Subsystem for Linux (WSL) on a Dell XPR 15 9520 laptop with 16 GB of RAM, demonstrating that the preservation process does not necessarily require specialized hardware. AdventureWorks provides a well-known benchmark with a complex but controlled relational schema, including numerous tables, constraints, and relationships, whereas HBR represents

a large-scale, real-world database with heterogeneous data types and historical depth. Together, these datasets allowed us to evaluate the DbSpec process across both synthetic benchmark data and production-grade archival data. Notice that most of the time is spent outside the interpreter itself, executing SQL or shell scripts and performing the SIARD extraction (using SIARD Suite). This is especially true when the database gets larger.

B. ROAE SHELL EVALUATION

This subsection evaluates the ROAE shell as a solution for the database recreation and future use, with particular emphasis on the non-interactive components of the workflow. The objective is to demonstrate that recovery procedures derived from preserved objects generated by DbSpec can be carried out in both current and future execution environments, and to provide an indication of the computational resources required.

To support large-scale recovery scenarios (iVM simulation might take large execution times for large databases) and to isolate functional aspects of the recovery process from resource constraints, these experiments were carried out on a high-performance server with Intel Xeon Platinum 8380 processors and 1007GB of RAM, running Linux Ubuntu 22.04.

These tests focus on the bootstrap-equivalent routines corresponding to the automated, non-interactive part of database recovery (see Sec. V-F). This consists of two main steps: (i) unzipping the preserved SIARD and transforming it into its SQLite representation, and (ii) loading this representation into the chosen database management system (SQLite).

Two execution scenarios are considered: a present-day environment based on the x86 architecture, and a future-use environment using the iVM abstraction. Table 3 show the execution times and memory requirements for the four databases tested and these two scenarios: the two processed in the previous experiments with DbSpec, as well as the other two additional samples from MySQl.

In the present-day scenario, tests have been executed directly on a standard x86 platform using a conventional file system. This means that all intermediate steps (decompressing SIARD, generating SQL, and loading it to SQLite) are performed on the file system on disk, taking advantage from of the native execution and the operating system optimized support.

In contrast, the future-use scenario is recreated today via the emulation of the iVM architecture on a x86 hardware. This setup mimics the long-term recreation environment that future developers are intended to build from formal specifications, in order to recover the information preserved on the film reel.

Note that in the iVM emulator, both the content of the preserved film reel, as well as the intermediate files generated (unzipped SIARD and SQLite SQL) need to reside entirely in the in-RAM IVMFS file system. This is why this

```
ivmfs:/work> tree db
db
+-- simpledb.roa
+-- simpledb.siard
+-- 01-encoding.roae
+-- 01-encoding.siard
`-- siard_testfiles
    +-- lobs
    | +-- field
    | | +-- field
    | | | `-- record0.txt
    | | +-- record0.bin
    | | `-- record0.flac
    | `-- record0.txt
    +-- sample.siard
    +-- sfdboe.siard
    +-- sfdbsakila.siard
    +-- sql1999.siard
    `-- sql2008.siard

ivmfs:/work> sqlite -- loadsiard db/01-encoding.siard

  [... This unzip the SIARD file, converts its
       content to SQL, and finally loads it into the
       SQLite engine embedded in the ROAEshell ...]

Converting to SQL ...
/work/db/01-encoding.siard opened
File 'ivmfs:/work/db/01-encoding.siard' found 59
     entries
 extracting: header/metadata.xml
[...]

SIARD version 2.2
Found 1 schemas:
 encoding: 5 tables, 31 rows, 90 cells

SQL file: 'out.sql' 47182 bytes (47.18KB)

ivmfs:/work> sqlite -- tables # for information only

[... tables of the load db must be shown here ...]

ivmfs:/work> roae load db/01-encoding.roae

Read 12 commands from ROAE file 'db/01-encoding.roae'


ivmfs:/work> roae list
----------------------
Command number #0
----------------------
Command:
      title = "Get a list of tables in the database"
      Parameters:
      Body:
            ANALYZE main; SELECT tbl, stat FROM
      sqlite_stat1 GROUP BY tbl ORDER BY tbl;

[...]

ivmfs:/work> roae menu

[... The ROAE menu appears ...]
```

FIGURE 10. ROAE shell interactive example mimicking the reel bootstrap actions (demo site: https://ivm.ac.uma.es/roaeshell).

scenario exhibits significantly higher memory consumption, particularly for large databases such as HBR. This behavior is intrinsic to the iVM emulation and should not be interpreted as a limitation of the preservation model itself, but rather as an indication of the resource requirements for future developers.

One potential improvement would be to pipeline the different bootstrap steps, thereby reducing the need to store intermediate files, and lowering the file system size requirements. However, such optimizations fall outside the scope of this work. The primary goal of these experiments is to validate the correctness and long-term viability of the iDA solution.

These experiments have shown the feasibility and effectiveness of the iDA solution across diverse databases,

TABLE 1. Features of the tested databases sorted by SIARD file size.

DB	schemes	tables	rows	cells	LOBs	SIARD size	SQLite size
HBR [14]	1	9	63086533	667512276	15425	922 MB	8.5 GB
AdventureWorks [15]	6	68	759240	6843103	149855	54 MB	136.4 MB
Employees [16]	1	6	3919015	16276090	0	40 MB	287.6 MB
Sakila [17]	3	21	52588	320168	2001	1.5 MB	5.1 MB

TABLE 2. DbSpec process execution time breakdown for the two realistic databases *HBR* [14] and *AdventureWorks* [15].

DB	Total time	Shell command time	SQL command time	SIARD extract time
HBR [14]	18407 sec. (05:06:47)	8588 sec. (46.7%)	2176 sec. (11.8%)	7460 (40.5%)
AdventureWorks [15]	130 sec. (00:02:10)	17.5 sec. (13.5%)	< 1 sec. (< 1%)	102.5 sec. (79%)

TABLE 3. x86 execution time and resident memory for bootstrap-equivalent SIARD-to-SQL action.

DB	Exec. time	x86 instructions	Max. resident RAM
HBR	930 sec. (00:15:30)	8.12 Ti	91 GB
AdventureWorks	18 sec. (00:00:18)	141.05 Gi	324 MB
Employees	24 sec. (00:00:24)	197.94 Gi	3 GB
Sakila	< 1 sec.	4.46 Gi	36 MB

TABLE 4. iVM execution time and resident memory for bootstrap-equivalent SIARD-to-SQL action.

DB	Emulator exec. time	ivm64 instructions	x86 instructions	Max. resident RAM
HBR	55674 sec. (15:27:54)	34.32 Ti	386.09 Ti	110.5 GB
AdventureWorks	900 sec. (00:15:00)	576.22 Gi	6.94 Ti	1.82 GB
Employees	854 sec. (00:14:14)	485.70 Gi	5.77 Ti	3.64 GB
Sakila	41 sec. (00:00:41)	17.38 Gi	298.42 Gi	719 MB

TABLE 5. x86 execution time and resident memory for the bootstrap-equivalent SQL-loading action.

DB	Exec. time	x86 instructions	Max. resident RAM
HBR	780 sec. (00:13:00)	5.40 Ti	8.33 GB
AdventureWorks	8 sec. (00:00:08)	55.63 Gi	103.88 MB
Employees	35 sec. (00:00:35)	235.14 Gi	289.11 MB
Sakila	< 1 sec.	2.72 Gi	9.92 MB

TABLE 6. iVM execution time and resident memory for the bootstrap-equivalent SQL-loading action.

DB	Emulator exec. time	ivm64 instructions	x86 instructions	Max. resident RAM
HBR	38016 sec. (10:33:36)	23.36 Ti	274.30 Ti	16.98 GB
AdventureWorks	403 sec. (00:06:43)	232.25 Gi	2.73 Ti	348.47 MB
Employees	3546 sec. (00:59:06)	2.38 Ti	30.72 Ti	6.85 GB
Sakila	37 sec. (00:00:37)	11.30 Gi	254.09 Gi	801.46 MB

including large-scale realistic ones, without making assumptions about future hardware or operating system support.

C. ROAE SHELL MICROEXPERIMENTS

In addition to full database scenarios, a set of micro-experiments were also conducted to test specific features of the ROAE shell. Among these features, the ability to manage user-defined data types (UDT), and handling both internal and external large objects (LOBs) have been tested with the samples tests provided in the SIARD repository [19]. Such samples contain several minimal SIARD archives designed to cover specific preservation features, and database standards. Also a synthetic SIARD/ROAE pair were used to check the correct behaviors of the UTF-8 encoding (default for SIARD) in the iDA solution.

All these micro-experiments complement the larger-scale evaluations by validating the robustness of the ROAE software. Their files are available in the ROAE shell demo site, https://ivm.ac.uma.es/roaeshell, under the `db` folder. Fig. 10 shows the transcription of converting and loading one of these SIARD micro-examples with its associated ROAE use cases.

VII. RELATED WORK

As mentioned earlier, the preservation process produces a digital object with all the information required

TABLE 7. A comparison of iDA with other database preservation solutions.

Feature	SIARD	DBML	CHRONOS	DBPTK	iDA: DbSpec/ROAE
Main goal	Standard format and Java toolset. Vendor-neutral, ISO-compliant snapshots (eCH-0165) for archival compliance and data longevity.	Open-source DSL. Human–readable schema, documentation and visual design, abstracting DB architecture from implementation.	Commercial software & enterprise decommissioning tool. Audit-proof (legacy systems), OAIS-compliant.	Open-source migration toolkit. Normalization and Mediation Layer between sources and archival formats with visualization and re-ingest.	Executable specification and methodology mainly for very long-term preservation.
Ability to Preserve Queries	Static/Textual. Preserves DDL[10] for routines and views as metadata.	Structural only. Captures relational definitions like foreign keys and indexes.	Static/Textual. Extracts logic (stored procedures, views) as XML/text for audit trails.	Static/Textual. Extracts behavioral information (Views, Triggers) into SIARD/DBML metadata.	Use-case preserved through the proposed Read-Only Access Engine (ROAE)
Metadata Support	Standardized. eCH-0165 structural metadata.	Semantic. Inline tags for human-readable docs.	OAIS-Compliant. Focus on audit trails/legal-provenance.	Curatorial. Post-extraction enrichment tools.	As much as SIARD; additional information can be stored in ROAE files
SQL Dialects	Major systems (Oracle, MS SQL Server, MySQL, PostgreSQL, DB/2, and MS Access). Limited native support for SQLite.	Agnostic (universal bridge) (PostgreSQL, MySQL, and SQL Server).	Broad. Most enterprise DBMS and legacy mainframes via JDBC/ODBC.	Extensive. Supports all major relational databases and legacy formats like MS Access; Plugins extension support.	Uses logical abstraction. Dialect-independent. Logical preservation of data structures and use-cases. Transparently managed underlying DBMS included in the ROAE shell.
Execution Environment	Java-based Desktop/CLI.	Web/CLI. (Node.js CLI tools, JavaScript, or browser-based visualizers like dbdiagram.io).	Java-based enterprise server.	Multi-platform. (Desktop GUI, enterprise web-based service, developer-centric Java library and CLI).	Specification only. Does not require a persistent runtime; Transformation logic to be executed by compatible preservation engines.
Long-term reproducibility	Standard-Based. Relies on the SIARD open format (XML/ZIP). Ensures data is readable by any XML parser.	As a plain-text DSL, is platform-independent and ensures database structures remain human-interpretable.	Proprietary/System-managed. System-independent XML formats, but proprietary solution.	Standard-Based. Leverages SIARD 2.0 and DBML to produce non-proprietary system-independent outputs.	Very high. Preservation verifiable and reproducible. Supported by abstract machine and preservation substrate (film).
Licensing	Free / Open Source (CDDL and GPL)	Open Source (MIT).	Commercial / Proprietary.	Open Source (LGPL/GPL).	Open Specification (github available).
Best Use Case	Small-to-medium archives (GLAM[11]).	Brainstorming and documenting schemas.	Large enterprise decommissioning.	Large-scale migrations into SIARD.	Complex, iterative preservation/audit; long-term preservation (film reel / iVM abstract machine).

[10]Data Definition Language
[11]Galleries, Libraries, Archives, and Museums

to be preserved, and it can be seen as a sequence of tasks [20], [21], [22]. One important standard for archival and data preservation is OAIS (Open Archival Information System) whose formalized version is the ISO 14721 reference model [18], [23]. In the OAIS model, information is packed into a Submission Information Package (SIP), which is the basis for creating an Archival Information Package (AIP). For future users, a Dissemination Information Package (DIP) is to be distributed, so that the preserved content can be retrieved.

Focusing on archival formats for the preservation of relational databases, the SIARD standard[9] [1], [24] plays an important role in this domain. It is a result of the collaboration between the Swiss Federal Archives and e-ARK [25]. As described, the three building blocks of SIARD are the XML storage, the Unicode encoding, and the use of URI universal location descriptors.

In addition to saving a database using markup languages, such as XML, two other major preservation approaches are migration and emulation [26]. With the migration method, the contents of a retired database system is transferred to a more modern and up-to-date system. The emulation method consists of simulating an old computing environment using virtualization technologies. This allows an outdated database to be used on a contemporary software that

[9]Latest version is SIARD 2.2.

emulates the original development environment, thereby facilitating its accessibility on modern systems. Emulation and virtualization are not a mere alternative to migration, but it might be an essential component of a robust long-term preservation flow [27], [28].

Both methods have the disadvantage of being software and architecture dependent. Approaches such as SIARD do not require specific hardware or software and are readable by both humans and machines, making it a sustainable format for preservation and storage purposes. However, when data is converted to XML format, certain interactive functionalities of the database, such as the query ability, are lost.

Other alternative proposals found in the literature include DBML [29], CHRONOS [30], and DBPTK (Database Preservation Toolkit) [31], although this last one is a SIARD-based solution. A comparison between these different solutions and iDA is shown in Table 7.

VIII. CONCLUSION

The medium- to long-term preservation of database systems, commonly after their decommissioning, poses significant challenges to which the Immortal Database Access (iDA) solution seeks to address. Based on the former Immortal Virtual Machine (iVM) architecture, iDA uses the SIARD format for preserving content on photographic film and introduces the DbSpec language to describe decommissioning processes and the ROAE concept to interact with the use cases. ROAE and SIARD are the preservation formats of the solution, to be generated from the rules specified in the DbSpec language.

As a means of accessing preserved data in the future, the ROAE shell has been introduced as an interactive console that allows future users to execute the predefined and preserved use cases. Such a shell is built based on the toolchain available for the iVM architecture, written in C/C++, with open source libraries. It can be compiled and executed in today's systems, but also it is designed to be preserved for future users using the film substrate. In this long-term preservation scenario, ROAE shell binaries for iVM are stored on the film along with the SIARD and ROAE files. They are intended to run after the film's bootstrap process, allowing the user to interact with the preserved database.

The development of the iDA solution has been thoroughly tested and made public in the repositories cited in the appendix.

APPENDIX

REPOSITORIES AND DEMO SITE

The documentation and source codes of the software developments described in this article, including the GCC C/C++ compiler for iVM and an iVM implementation, are available in the following GitHub repositories.

- DbSpec: https://github.com/immortalvm/dbspec
- ROAE shell: https://github.com/immortalvm/ROAEshell
- SIARD to SQLite-compliant SQL converter: https://github.com/immortalvm/siard2sql
- iVM Documentation: https://github.com/immortalvm/ivm-doc
- iVM GCC CC/C++: https://github.com/immortalvm/ivm-compiler
- iVM emulator: https://github.com/immortalvm/yet-another-fast-ivm-emulator
- iVM in-RAM filesystem: https://github.com/immortalvm/ivm-fs

The ROAE shell demo site can be accessed via the following URL:

- https://ivm.ac.uma.es/roaeshell

ACKNOWLEDGMENT

The authors would like to thank Bjarte M. Østvold from Norwegian Computing Center, for his contributions to the iDA Project and DbSpec.

REFERENCES

[1] H. Bruggisser, "SIARD format specification 2.0," eCH: E-Government Standards, Swiss, Basel, Switzerland, Tech. Rep. eCH-0165, 2015.
[2] I. Rummelhoff, E. Gutiérrez, T. Kristoffersen, O. Liabø, B. M. Østvold, O. Plata, and S. Romero, "An abstract machine approach to preserving digital information," *IEEE Access*, vol. 9, pp. 154914–154932, 2021.
[3] R. M. Stallman, "GNU compiler collection internals (for GCC version 12.2.0)," Tech. Rep., 2022. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-12.2.0/gccint.pdf
[4] C. Vinschen and J. Johnston, "The Red Hat newlib C library," Tech. Rep., 2021. [Online]. Available: https://sourceware.org/newlib/
[5] J. Bennett. (2010). *Howto Porting Newlib: A Simple Guide*. Accessed: Mar. 1, 2021. [Online]. Available: https://www.embecosm.com/appnotes/ean9/ean9-howto-newlib-1.0.html
[6] I. Rummelhoff, T. Kristoffersen, and B. Østvold, "Reproducible preservation of databases through executable specifications," *Int. J. Digit. Curation*, vol. 19, no. 1, p. 16, Jun. 2025.
[7] D. R. Hipp. (2025). *SQLite*. [Online]. Available: https://www.sqlite.org/
[8] *Library of Congress—Recommended Formats Statement*. Accessed: Feb. 14, 2026. [Online]. Available: https://www.loc.gov/preservation/resources/rfs/data.html
[9] (2025). *Swiss Federal Archives*. Accessed: May 10, 2025. [Online]. Available: https://www.bar.admin.ch/bar/en/home/archiving/tools/siard-suite.html
[10] L. Thomason. (2025). *TinyXML2: A Simple, Small, Efficient, C++ XML Parser*. Accessed: Jul. 24, 2025. [Online]. Available: https://github.com/leethomason/tinyxml2
[11] J.-l. Gailly and M. Adler. (2024). *ZLIB: A Massively Spiffy Yet Delicately Unobtrusive Compression Library*. Accessed: Jul. 24, 2025. [Online]. Available: https://www.zlib.net/
[12] E. Gutierrez, S. Romero, and O. Plata. (2024). *The IVM Filesystem Generator*. [Online]. Available: https://github.com/immortalvm/ivm-fs
[13] SQLite. (2025). *Prepared Statement Object (Sqlite C Interface)*. Accessed: Sep. 15, 2025. [Online]. Available: https://www.sqlite.org/c3ref/stmt.html
[14] Norwegian Computing Center. (2026). *Historisk Befolkningsregister (HBR)*. Accessed: Jan. 1, 2026. [Online]. Available: https://nr.no/en/projects/the-norwegian-historical-population-register/
[15] Microsoft. (2026). *Adventureworks Sample Databases*. Accessed: Jan. 1, 2026. [Online]. Available: https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure
[16] MySQL. (2026). *Employee Sample Database Documentation*. Accessed: Jan. 1, 2026. [Online]. Available: https://dev.mysql.com/doc/employee/en/
[17] MySQL. (2026). *Sakila Sample Database Documentation*. Accessed: Jan. 1, 2026. [Online]. Available: https://dev.mysql.com/doc/sakila/en/
[18] OAIS. (2025). *OAIS Reference Model Site*. Accessed: May 10, 2025. [Online]. Available: http://www.oais.info

[19] Swiss Federal Archives. (2025). *SIARD API: Test Files*. [Online]. Available: https://github.com/sfa-siard/SiardApi/tree/cc65c659b5a280f0bc4de492d355e562c916eaa5/src/test/resources/testfiles
[20] K. Thibodeau, "Overview of technological approaches to digital preservation and challenges in coming years," in *The State of Digital Preservation: An International Perspective*, 2005, pp. 4–31. [Online]. Available: https://www.clir.org/pubs/reports/pub107/thibodeau/
[21] S.-J. Cha, Y. J. Choi, and K.-C. Lee, "Development of preservation format and archiving tool for the long-term preservation of the database," in *Proc. 9th Int. Conf. Ubiquitous Inf. Manage. Commun.*, 2015, pp. 1–4.
[22] J. C. Ramalho, B. Ferreira, L. Faria, and M. Ferreira, "Beyond relational databases: Preserving the data," *New Rev. Inf. Netw.*, vol. 25, no. 2, pp. 107–118, Jul. 2020.
[23] ISO. (2025). *ISO 14721:2025 Space Data System Practices–Reference Model for an Open Archival Information System (OAIS)*. [Online]. Available: https://www.iso.org/standard/87471.html
[24] SIARD. (2024). *Software Independent Archiving of Relational Databases (SIARD) Download Page*. Accessed: Dec. 15, 2024. [Online]. Available: https://dilcis.eu/content-types/siard
[25] J. Anderson, K. Aas, D. Anderson, and A. C. Wilson, "The E-ARK project: An introduction to the European archival records and knowledge preservation project," *New Rev. Inf. Netw.*, vol. 25, no. 2, pp. 83–92, Jul. 2020.
[26] K. H. Lee, O. Slattery, R. Lu, X. Tang, and V. McCrary, "The state of the art and practice in digital preservation," *J. Res. Nat. Inst. Standards Technol.*, vol. 107, no. 1, p. 93, Jan. 2002.
[27] R. Appuswamy and V. Joguin, "Universal layout emulation for long-term database archival," in *Proc. 11th Conf. Innov. Data Syst. Res.*, Jan. 2020, p. 8. [Online]. Available: https://vldb.org/cidrdb/papers/2021/cidr2021_paper30.pdf
[28] A. Acker, "Emulation practices for software preservation in libraries, archives, and museums," *J. Assoc. Inf. Sci. Technol.*, vol. 72, no. 9, pp. 1148–1160, Sep. 2021.
[29] DBML. (2025). *Database Markup Language (DBML)*. Accessed: Sep. 15, 2025. [Online]. Available: https://dbml.dbdiagram.io/home
[30] A. Lindley, "Database preservation evaluation report: SIARD vs. CHRONOS," in *Proc. Int. Conf. Preservation Digital Objects*, vol. 10, 2013, pp. 29–38. [Online]. Available: https://www.digipres.org/publications/ipres/ipres-2013/papers/ipres-2013-proceedings-of-the-10th-international-conference-on-p/
[31] J. C. Ramalho, L. Faria, H. Silva, and M. Coutada, "Database preservation toolkit: A flexible tool to normalize and give access to databases," in *Proc. DLM Forum-7th Triennial Conf.*, 2014, p. 63.

ELADIO GUTIÉRREZ received the M.Sc. and Ph.D. degrees in telecommunication engineering from the University of Málaga, Spain, in 1995 and 2001, respectively. Since 2003, he has been an Associate Professor with the Department of Computer Architecture, University of Málaga.

IVAR RUMMELHOFF received the Ph.D. degree in mathematical logic. He is currently a Senior Research Scientist with Norwegian Computing Center, focusing on digital transformation. He has a background in the software industry.

SERGIO ROMERO received the M.Sc. and Ph.D. degrees in computer science from the University of Málaga, Spain, in 1996 and 2000, respectively. Since 2003, he has been an Associate Professor with the Department of Computer Architecture, University of Málaga.

THOR KRISTOFFERSEN received the Ph.D. degree in computer science from the University of Oslo, in 1998. Since then, he has been with Norwegian Computing Center, where he is currently a Senior Research Scientist.

JOSÉ A. TIRADO-DOMÍNGUEZ received the M.Sc. degree in telecommunication engineering from the University of Málaga, Spain, in 2001, where he is currently pursuing the Ph.D. degree in computer science. He was with Tedial, in 2006, where he has been the Research and Development Director, since 2016.

MARIA DEL CARMEN LÓPEZ received the M.Sc. degree in automatic and electronic industrial engineering from the University of Málaga, Spain, in 2008. She joined Tedial, in 2006, where she is currently the Head of Innovation.

OSCAR PLATA received the Ph.D. degree in physics from the University of Santiago de Compostela, Spain, in 1989. He started as an Assistant Professor with the University of Santiago de Compostela, where he became an Associate Professor, in 1990. He moved to the University of Málaga, in 1995, where he became a Full Professor with the Computer Architecture Department, in 2002. His research interests include high performance computing and parallel architectures. He was an Associated Editor of IEEE TRANSACTIONS ON COMPUTERS, from 2015 to 2019.