

SQL Access and IBM DRDA A Comparison in a Multi-Vendor Setting

Scott Newman
Database Systems Engineering
Digital Equipment Corporation
55 Northeastern Blvd., NUO1-1/B09
Nashua, New Hampshire, 03062-1260
e-mail: Newman@broke.enet.dec.com

Jim Gray
San Francisco Systems Center
Digital Equipment Corporation
455 Market St., 7th Fl.
San Francisco, California, 94105-2403
e-mail: JimGray@sfbay.enet.dec.com
December 1991

Introduction

This paper¹ compares the IBM Distributed Relational Database Architecture (DRDA²) and the set of standards that form the definitions of the SQL Access Group. The comparison is done in a multi-vendor setting, in which client and server are often running on different hardware or software platforms, and are supplied by different vendors. At this point in the evolution of *heterogeneous* database interoperability, it is important that a comparison be available because of the competitive light in which these two approaches are viewed.

This paper is divided into four sections. The executive summary and overview sections provide summary and background information. The main body of the paper divides the comparison into technical and non-technical sections.

Terminology

Many standards and architectures are referred to throughout this paper. They have long and involved names or multi-letter acronyms, that can be either tiresome or confusing.

To enhance readability, the name *SQL Access* is used here to represent either the SQL Access Group or one of the ISO, ANSI, or X/Open definitions upon which the SQL Access definitions are based. Similarly, the term *DRDA* is used to represent either the DRDA architecture itself or one of the

related IBM architectures. Please refer to Section 3 for more detail on how these standards and architectures interrelate.

Executive Summary

SQL Access and DRDA are two very different architectures for client-server database interoperability. They have diverse origins; one is the result of a collaborative effort by many companies, the other a result of an extensive effort by a single company. Not surprisingly, each approach is best suited to the environment in which it was born.

The major differences between SQL Access and DRDA are:

DRDA is owned by IBM; SQL Access is owned by a consortium of 42 vendors and users. (Section 4.2)

SQL Access is based on international standards; DRDA is based on IBM architectures. (Section 3)

In the SQL Access design, all database servers support the same SQL syntax, semantics and datatypes, and they share a common message encoding format. This is called the *common-subset, canonical form approach*. (Sections 5.1 and 5.3)

In DRDA, each client and server speaks its own dialect of SQL and data encodings. This is called the *anything-goes, receiver-makes-it-right approach*. (Sections 5.1 and 5.3)

An existing DRDA network will be perturbed by the introduction of a new type of client or server. If a new type of server is added, existing client applications that access it need to use an SQL dialect supported by the new server. If the new client or server uses a new data encoding format, existing clients or servers accessing it must add support for the new encoding format. This approach makes heterogeneous operability expensive difficult to manage. (Sections 5.1 and 5.3)

SQL Access emphasizes application portability between heterogeneous systems; DRDA provides a form of application portability that permits applications to be moved between client platforms provided that the type of server being accessed remains the same. (Section 5.4)

A SQL Access client or server implementor has

¹A condensed form of this paper appeared as 'Which Way to Remote SQL?' in *Database Programming and Design*, V.4.2, Dec. 1991, pp. 46-54.

²IBM DRDA should not be confused with ISO Remote Database Access, RDA. They are unrelated.

the support of software tools and a growing body of expertise. A DRDA client or server implementation requires significantly more development effort due to the lack of software tools, the protocol complexity, the message encoding model and the required support for packages. (Sections 4.4, 5.3, 5.9, and 5.12)

DRDA supports precompiled SQL statements stored at the server in packages. By using packages, the execution performance at the server can approach the performance of the local case. (Section 5.12)

SQL Access client applications have more flexibility when selecting servers than do DRDA client applications because the same SQL variant is provided by all servers. DRDA client applications that are tied to a particular server type may make use of all of that server's features. (Section 5.1)

SQL Access uses OSI networking; DRDA uses IBM's proprietary networking (SNA). (Section 5.6)

In summary, DRDA is a remote database access protocol defined by IBM. SQL Access is based on existing or proposed international standards. DRDA is oriented toward intra-IBM interoperability, SQL Access is focused on multi-vendor interoperability. Heterogeneous portability combined with demonstrated interoperability, suggest SQL Access will become the prevalent heterogeneous database interoperability solution.

Table 1
DRDA and SQL Access Contrasted

Issue	SQL Access	DRDA
definer goals	consortium heterogeneous portability and interoperability	IBM remote access to IBM database servers
approach	common subset	anything-goes receiver-makes-it-right
protocols for m clients and n servers	$n + m$	$n \times m$

Overview

The Players

Several standards bodies are referenced throughout this paper. In order to distinguish them, the following definitions are offered:

International Standards Organization (ISO) is an international standards body comprised of national standards bodies. The Open Systems Interconnection (OSI) Model is defined by ISO standards, as is the SQL database language.

American National Standards Institute (ANSI) is the national standards body representing the United States to ISO.

X/Open is an independent, international systems consortium of vendors. Its focus is portability and practical implementation of open systems.

SQL Access

The SQL Access Group is a consortium of 42 member companies that was formed in 1989. Its members include almost all major vendors of database software and tools, as well as some companies that are end-users of such products. Although IBM is a member of X/Open and is very active in the relevant ANSI and ISO standards committees, IBM has not yet joined the SQL Access Group.

The focus of the SQL Access Group is to accelerate existing standards efforts and prove their viability through prototyping. The group's efforts have resulted in a number of submissions to the ISO Remote Database Access (RDA), and ISO and ANSI SQL2 committees. Most of these proposals have been incorporated into the applicable standards.

The SQL Access specifications are published by X/Open. To-date, the group has produced two specifications:

An application programming interface (API) specification [1] that defines an embedded database language specification, based on the ANSI and ISO SQL definition known as *SQL-89* [3].

A formats and protocols (FAP) specification [2] for client-server communication, based on the ISO Remote Database Access SQL Specialization [5, 6].

The SQL Access API specification defines an embedded SQL language based on SQL-89 [3]. In order to support the client-server model, language elements from the SQL2 specification were adopted [4]. Some of these language elements, such as those used for client-server association³ management, were defined by SQL Access, presented to the ANSI/ISO standards committees, and were adopted as part of SQL2.

The current SQL Access FAP specification is a short *differences* document from specific versions of the ISO RDA Generic and SQL Specialization specifications [5,6]. In addition to the clarifications, implementor's agreements and

³A client-server association is called an *SQL-connection* in SQL2.

limits, this specification also contains the change proposals that were submitted to the ANSI RDA committee and later (mostly) adopted by ISO RDA. SQL Access specifications augment the standards on which they are based. They define areas that the underlying standards consider to be *implementor-defined*. For example they specify lower limits on implementor choices so that portable applications can be written, and so that systems working within these limits can interoperate. These implementor agreements are an established part of the standards process.

In X/Open terminology, the API specification is a *preliminary specification*, which means it is fairly stable. The FAP specification is considered by X/Open to be a *snapshot* specification: It describes work in-progress that is worthy of dissemination.

DRDA

DRDA is an IBM-owned architecture that addresses database interoperability. The initial focus of DRDA was to provide a vehicle for interoperation between IBM's four relational database managers. More recently, IBM has provided DRDA specifications and seminars to other companies, so DRDA can be used for multi-vendor interoperability as well.

The DRDA specification [7] defines a model for client-server interaction based on several other IBM architectures, including SNA Logical Unit type 6.2 (LU6.2). Many aspects of the model are defined in detail, including such operational features as interaction with SNA network management.

DRDA draws upon the following IBM architectures and extends them as required:

SNA Logical Unit type 6.2 (LU6.2)

Distributed Data Management Architecture (DDM)

SNA Management Services Architecture (MSA)

Formatted Data Object Content Architecture (FD:OCA)

Character Data Representation Architecture (CDRA)

Two more advanced *levels* of DRDA provide an architectural direction for DRDA's future.

Current Status

The SQL Access Group completed its *Phase I* effort in July 1991, culminating in a public multi-vendor interoperability demonstration of 19 client and server prototypes. At that time, the specifications became available through X/Open. The group is

now beginning *Phase II*, which will include conformance testing, the use of TCP/IP, a call-level programming interface, and persistent (precompiled) SQL statements stored at servers. Future phases may address multi-server transactions, stored procedures, large objects, and enhanced security.

To advance the FAP specification beyond the snapshot level, an effort is underway to align it with the recently progressed Draft International Standard version of the RDA specification. The FAP implementors' agreements are also being coordinated with the RDA SIG at the National Institute of Standards and Technology (NIST) OSI Implementors' Workshop. The SQL Access FAP implementors' agreements were adopted by the NIST RDA SIG as the core of its base document.

We believe products based on SQL Access will appear late in 1991. SQL Access gateways to IBM database servers have been demonstrated in addition to the nine servers and ten clients at the July SQL Access interoperability demonstration.

DRDA clients and servers are currently being implemented by several IBM relational data managers. Recent IBM announcements state that DRDA will be used for interoperability in product releases in March 1992. IBM has hosted two workshops for companies interested in learning about DRDA. In addition, nine companies have announced an intention to provide DRDA implementations in order to access data at IBM servers. A number of these companies are also SQL Access members, some of which also have working SQL Access client and server prototypes.

Non-Technical Differences

There are a number of differences between SQL Access and DRDA that are non-technical in nature. Some differences have an impact on the practical aspects of product development; others affect how the specifications will evolve.

Types Of Standards

There are two types of public standards:

A *de facto standard* is created when one company's product dominates an area to such an extent that other companies follow with their own implementations.

A *de jure standard* is established by a standards organization through a formal process. International computer vendors must provide products that conform to the applicable *de jure* standards in order to satisfy the procurement criteria of governments and industries in many

countries.

SQL Access' goal is to advance *de jure* standards by first prototyping designs, and then proposing incremental changes to existing standards bodies. As these proposals are incorporated in ISO standards and implementor agreements, they become *de jure* standards.

DRDA is an IBM architecture that might become a *de facto* standard after some time, if other companies decide to implement it. If this happens, vendors will be compelled to implement both approaches, or at least to implement a SQL Access to DRDA gateway.

Ownership

The issue of ownership of specifications is at the core of many non-technical issues. Ownership ultimately dictates who controls the content of a specification. The specifications that form the SQL Access definitions are controlled either by national and international standards bodies or by consortia. A company that wishes to provide input to the specifications or influence their direction is free to join any or all of the standards bodies, and work to affect the standards.

DRDA is owned by IBM. Its specifications are copyrighted by IBM. IBM has indicated that it will license DRDA to interested parties for a nominal fee.

Change Process

SQL Access and DRDA are evolving technologies. The current DRDA specification describes the first of three architectural *levels*, termed *remote-unit-of-work*. The recently published SQL Access specifications are the result of the first *phase* in SQL Access' evolutionary approach to database interoperability.

The direction of the SQL Access effort is determined through a committee process in which member companies are free to make proposals. Technical changes to specifications are carried to one of the technical working groups by member companies. Each member company is entitled to one vote on each committee in which it participates.

When appropriate, SQL Access submits change proposals to the ANSI X3H2 (SQL) and X3H2.1 (RDA) committees. Such proposals are submitted by standards committee members representing their companies, and voted on using the normal ANSI committee rules. Many of the company representatives on the SQL Access technical committees also represent their companies on the

corresponding ANSI committee.

The DRDA change process is managed by an internal IBM architecture committee with representation from the four major IBM relational database products (DB2, SQL/DS, OS/400 and OS/2 EE Data Manager). IBM will probably provide a mechanism through which interested companies can participate in DRDA's evolution. However, it is unlikely that the procedure for approving architecture changes will approach the equity of the *one company, one vote* forum of SQL Access and the national standards bodies.

In addition to its in-house DRDA effort, IBM is an active member of the RDA and SQL2 committees at ANSI and national standards bodies in other countries. A number of improvements and additions to RDA and SQL2 are due to IBM – some of which parallel corresponding facilities in DRDA.

Implementation Difficulty

The implementation of either a DRDA or SQL Access client or server is a very significant undertaking. Bringing an architecture or a standard from the paper stage to a working implementation is a long and arduous process – particularly in the multi-vendor interoperability setting.

The following sections examine aspects of SQL Access and DRDA that have a significant impact on implementation difficulty and cost. The focus is on the implementation of the client or server facility itself, not the application program that uses them.

Software Tools

Software tools can greatly accelerate an implementation effort. If a number of separate implementations use the same tools, some reduction in the interoperability testing effort may also be realized.

SQL Access message formats are defined by an ASN.1 module that is available through electronic mail. A number of message format compilers are available that automatically generate complete sets of message encoding and decoding routines. This alleviates much of the tedious, error-prone programming associated with implementing communications protocols.

No corresponding tools are available for DRDA implementations, so developers must hand-code these routines, and many tables of constant values. This lack of DRDA tools increases development costs and the probability of programming and interoperability problems.

Available Expertise

When developing an implementation from an architecture or standards specification, no matter how well-written, there are invariably points of confusion. It is important for implementors to have access to one or more experts in order to gain insight into what the specification really means.

The DRDA manuals are steeped in terminology from varied IBM environments, such as: SNA for LU6.2, AS/400 for DDM, and MVS for the encoding architecture. Although IBM is offering DRDA classes and testing facilities, all the designers of DRDA are architects or key engineers at IBM. Few engineers outside of IBM have a grasp of the diverse and elaborate IBM architectures required to understand and implement DRDA.

In the standards arena, each participating company has its own representative who can either answer detailed questions on a standard, or locate someone who can. The formation of the SQL Access group has brought together a large body of individuals comprised of both standards committee participants and developers (sometimes the same person). These individuals worked together to increase their expertise – then they put it to the test by prototyping implementations.

In addition to the expertise within SQL Access and the associated standards committees, there is an ever-growing pool of OSI application expertise in both the industrial and academic realms.

Technical Differences

There are many technical differences between SQL Access and DRDA. Some differences, in areas such as language and catalog tables, affect the application programmer's ability to write portable applications. In other areas, such as data value and message encoding, client and server implementation effort and complexity are affected.

Language

The approach to database language is an area of significant difference between DRDA and SQL Access. Both use SQL as their database language, but the variants of SQL and the way in which they are used by a client application are dramatically different.

SQL Access uses a *common subset* approach in which a single language is used. The single, standard SQL variant gives client applications uniform SQL syntax and semantics on any conforming server from any source. This approach relieves the portable applications developer of the task of finding a common subset among the SQL

dialects supported by heterogeneous servers. It also permits the selection of the target server to be deferred until run-time.

In the *common subset* approach, client application tools such as precompilers are able to provide direct support for the single SQL variant used. The development and packaging of end-user tools, such as query tools and fourth-generation languages, is simplified because tools can be written using a single, standard SQL variant that is supported across all servers.

DRDA takes an *anything-goes* approach in which any variant of SQL may flow from a client to a server. The SQL statements invoked at a particular server must use the variant of SQL supported by the server's data manager. This means that the SQL syntax, semantics and SQL data types supported by a specific server are exposed to the client application.

In DRDA's design, a client applications developer must be aware of the semantic and syntactic idiosyncrasies of the server, as well as specifics of SQL data types it supports. This means that the application developer must have a working knowledge of each type of data manager at each DRDA server that might be accessed.

Early in the development cycle, the DRDA client application developer must select the type of server it plans to access, in order to take data manager-specific details into account. This limits the freedom of the application user to choose a server with a different data manager at run-time. Portable application developers must search-out a common SQL subset supported across all of the servers to which access is planned.

In order to support the *anything-goes* model, DRDA client application tools must be extremely tolerant of the SQL syntax that they allow. If a precompiler is one of the tools for an existing data manager, it will likely need to provide two very different modes of operation. In its *normal mode*, the precompiler would support only its native SQL variant, and provide all of its usual application support and syntax checking. In an *anything-goes mode*, the precompiler would perform minimal, if any, SQL syntax checking in order to permit another variant of SQL to pass through it on its way to the server.

IBM's SAA provides assistance to DRDA applications that wish to be less dependent on a particular server type. SAA SQL [8] provides a definition of SQL intended to work across all SAA compliant data managers. While the SAA SQL definition goes a long way toward pulling together

the SQL variants supported by the four IBM SAA data managers, there are areas (e.g., SQL data type support) where data manager specifics still show through. More significantly, this SAA definition is primarily based on existing IBM SQL variants. It is not a multi-vendor specification, such as an international standard, and it lacks certain key components like a standard database catalog.

DRDA and SQL Access both permit the client application to use SQL extensions that are supported by the server. DRDA permits this inherently through its *anything-goes* database language approach. SQL Access provides an *escape clause* mechanism through which many non-standard extensions can be supplied without perturbing database managers that do not support the extensions.

Static (embedded) and dynamic SQL are supported by both SQL Access and DRDA. SQL Access' support for dynamic SQL is based on SQL2, but it is an extension to RDA. The RDA SQL Specialization does not yet address dynamic SQL, which is absent from SQL-89. It is expected that the SQL Access extensions to RDA for dynamic SQL will be proposed as enhancements to RDA.

In summary, SQL Access applications use a single SQL definition to access many different server implementations. In this environment, a client application that accesses, for example, three different servers can use a single variant of SQL throughout. DRDA applications use the SQL variant that is supported by the target server – which must be selected at development time. In the DRDA environment, a client application accessing three different servers could contain three different variants of SQL within the same program.

Catalog Tables

Catalog Tables, or Schema Information Tables, provide *metadata* that describes the data managed by a particular server. Client applications query the catalog to learn what tables exist and are accessible to a particular user and to obtain datatype information on specific columns within those tables. The existence of standard catalog tables is particularly important to decision-support and *ad hoc* query tools that rely on the catalog information to learn about the user's database.

SQL Access defines a set of catalog tables that have standard attributes and values. These tables are based on the corresponding SQL2 definitions. Additional server information is provided that augments standard information provided by SQL2 catalogs.

DRDA (or SAA) takes the *anything-goes* approach to catalog tables. It does not define a set of standard catalog tables. Applications obtain metadata information by issuing queries against the catalog tables that are provided by the server's underlying data manager. These tables vary significantly from data manager to data manager. For example, the catalog tables provided by the four IBM relational data managers are quite different.

The lack of standard catalog tables requires that the client application be aware of the data manager variant at a particular server. In addition, the client application must know the structure of the catalog information provided by each type of data manager that is likely to be accessed.

Message/Data Value Encoding

SQL Access and DRDA have dramatically different approaches to encoding protocol information and data values. One fundamental difference is that the SQL Access approach uses existing OSI standards, while DRDA is based on IBM architectures. The significant differences are summarized by Table 2.

SQL Access uses ISO Abstract Syntax Notation 1 (ASN.1) to define the messages that are used for communication between client and server. These definitions are independent of the transfer syntax (encoding) that is actually used when the messages are sent over the communications network.

The transfer syntax used for SQL Access messages is specified by the ISO Basic Encoding Rules (BER) for ASN.1 [10]. BER uses a Type/Length/Value triplet to convey a value. The value itself is represented in a well-defined, platform-independent, canonical form. The client and server must both convert to and from the canonical form, but each need only provide a single set of data conversion functions in order to do so.

Table 2
Encoding Characteristics

	SQL Access	DRDA
Encoding Strategy	canonical form	receiver-makes-it-right
Abstract Syntax	ISO ASN.1	DDM and FD:OCA
Transfer Syntax	ISO BER for ASN.1	DDM and FD:OCA
Negotiated?	Yes	No

In SQL Access, the choice of transfer syntax used is negotiated on a per-connection basis. The negotiation mechanism is provided by the OSI Presentation Layer, through which client and server can indicate their preferences and select accordingly.

DRDA uses Level 3 of IBM's Distributed Data

Management (DDM) Architecture to define the abstract syntax and encoding for commands and responses that flow between client and server. The abstract syntax and encoding for data and metadata are encoded using FD:OCA. FD:OCA is an IBM architecture that is also used in compound document architectures.

Message descriptors are provided in a multi-level scheme that eventually specifies the encoding format for the actual data values. Data values are encoded using one of three currently defined formats. These encoding formats are the native formats used on IBM's most popular platforms: System/370, AS/400 and Intel 80x86. IBM or another party could add additional encoding formats through a process administered by IBM. For example, IBM may choose to add a fourth encoding format in order to accommodate its RISC-based systems in DRDA.

The responsibility for data conversion is split between client and server in DRDA. At the time of protocol initiation, each side specifies a single encoding format it intends to use for the duration of the interaction. No negotiation occurs, but either side can reject the other's choice by refusing to participate. If the sender's encoding format is accepted, the receiver (client or server) of data must perform the conversion between the sender's data format and its native platform format, unless the receiver has the same native data types. This scheme is called *receiver-makes-it-right*.

DRDA's *receiver-makes-it-right* approach to data encoding minimizes data conversions and consequent loss of information. If conversion is needed, DRDA spreads the burden equally between client and server. Unfortunately, this scheme requires both sides to implement at least $n-1$ complete sets of data conversion routines for n encoding formats. If none of the n encoding formats matches a particular implementation platform's, the full n sets of conversion routines will be required, and at least one set of conversion routines will be required for data that is sent.

The *receiver-makes-it-right* approach has a benefit when dealing with character data. Character conversions that use an intermediate character set are prone to information loss. DRDA's approach eliminates information loss by removing unnecessary conversions. Although X/Open constrains character set support to conforming clients and servers, the SQL Access FAP supports diverse character sets and the minimization of character set conversions. Further enhancements to

character set support in ISO RDA are expected to be included in SQL Access as it aligns itself with the more recent version of RDA.

A natural language analogy illustrates the difference between the *canonical form* and the *receiver-makes-it-right* approaches. The canonical form approach corresponds to the universal language *Esperanto*, which is not the native language of any country in the world. Anyone wishing to use Esperanto would have to learn it, but if we all learned this one language we could speak with anyone else in the world.

The *receiver-makes-it-right* approach corresponds to a world in which many languages exist, and in which individuals always speak their native language. When individuals from different countries interact, each one speaks his native language, but he must learn the other's language in order to understand what others say. Each time a speaker of a different language is added, all participants must learn another language.

The costs and benefits of Esperanto and *receiver-makes-it-right* are fairly clear. If more than two languages are spoken, there is much less learning required if a single, standardized language is used by all parties concerned. These economies are the key rationale for standards.

In summary, SQL Access and DRDA have fundamentally different approaches to message and data value encoding. They both minimize or eliminate any information loss. While the ASN.1/BER approach almost always requires both client and server to convert a particular data value, one set of data conversions need be implemented by a particular client or server. The DRDA *receiver-makes-it-right* approach, is efficient among similar platforms, but in a heterogeneous network many sets of conversion routines must be implemented by each client and server. In addition, the realities of development resources and the administrative and synchronization issues associated with introducing support for new encodings in product releases, is likely to result in one or two encodings becoming the *de facto* encodings for DRDA.

Application Portability

Application portability is a main goal of SQL Access (interoperability is the other goal). SQL Access adopted an application-level definition for embedded SQL as its portability interface. It may be used by an application on any database platform conforming to the SQL Access API specification – which is X/Open's SQL specification and is a subset of the SQL2 language. This approach permits application portability across all

conforming client platforms, independent of whether the same or a different server is accessed.

DRDA provides a different type application portability. Application programs may be ported to different DRDA clients, provided that the application does not change the type of data manager that it is accessing at a DRDA server. For example, a CICS COBOL application that uses DB2 embedded SQL can continue to access DB2 remotely from an OS/2 DRDA client without any changes to its SQL-related parts. The DB2 variant of SQL, DB2 catalogs, and DB2 data types are identical when viewed remotely from the new client platform. This type of portability is one of the key motivations behind DRDA's *anything-goes* approach to language.

Diagnostics

SQL Access and DRDA both provide the application program with status information upon completion of each SQL operation. This is accomplished using both the SQLCODE and SQLSTATE mechanisms for return status codes.

The SQLSTATE return codes provided by SQL Access and DRDA are both based on the SQL2 SQLSTATE definition. SQL Access provides a standard set of SQLCODE return codes that are also SQL2-based. DRDA does not provide a standard set of SQLCODE return codes. Consistent with its *anything-goes* model, DRDA returns the SQLCODE provided by the underlying data manager at the server.

Network Requirements

SQL Access and DRDA both assume a particular communications network environment. SQL Access, being an early implementation of ISO RDA, is based on the OSI Reference Model and assumes the OSI addressing and naming structure. Only the most basic capabilities of the Session and Presentation Layers are employed for client-server communication.

Without significantly perturbing its formats and protocols, SQL Access could be adapted to run over any communication network that provides end-to-end, full-duplex, virtual circuit-type connections. TCP/IP and DECnet peer-to-peer communications are examples of popular network environments that provide the services needed by SQL Access. SQL Access is well into an effort to specify how database interoperation is achieved over a TCP/IP network.

DRDA uses SNA LU6.2 for client-server communications. The DRDA protocol description uses LU6.2 terminology. In addition, protocol flows

are described in terms of the LU6.2 verbs, and associated information, that are used at each step in an exchange.

Table 3

Network Requirements

	SQL Access	DRDA
Network Environment	ISO OSI	IBM SNA
Upper Layer Protocol	ACSE and Presentation	IBM LU6.2
Security	SQL Access and ACSE	IBM LU6.2
Duplex	Full	Half

The intimate relationship between DRDA and LU6.2 makes it difficult to determine clearly DRDA's dependence on LU6.2. For example, it is not evident if the LU6.2 naming, verbs and protocols could be mapped to similar entities in another network environment.

Network Management

DRDA defines specific *alerts* that are to be generated by a client or server upon detection of certain error conditions. Alerts are an SNA-specific mechanism for notifying a network control center that a specific problem has occurred. They are a useful problem-determination tool for production SNA environments.

An emerging OSI network management-related standard specifies a mechanism for OSI networks that is similar in function to alerts. At present, SQL Access has not utilized this capability.

Security

SQL Access and DRDA provide security at a number of levels. Both provide authentication based on passwords. Additional security is provided by the underlying networks.

Within SQL, a GRANT-REVOKE authorization model is used regulates client access to SQL objects.

Protocol/Message Complexity

It is difficult to assess the complexity of sets of messages and protocols in an objective manner. Complexity, in this case, affect the client and server development in ways such as: developer learning time, implementation time, implementation difficulty and the chances for successful implementation of a client or server. Two aspects of protocol and message complexity are examined: message content and encoding, and request chaining and asynchrony.

Message Content and Encoding

The message contents and encodings used by SQL Access are defined independently because of the

clean separation of abstract and transfer syntaxes. This separation is an aid to learning and dealing with the SQL Access formats and protocols because it permits one aspect to be focused upon, while putting the other aspect temporarily aside.

SQL Access messages are specified by an ASN.1 module that may be input to an ASN.1 compiler to automatically generate programs that encode and decode messages. This automatic generation is a tremendous jump-start for a client or server implementation effort. The semantics associated with the message content are described in the ISO RDA specifications referenced by SQL Access, with additional assumptions or implementor's agreements where required.

By contrast, DRDA combines the message content with the encoding. The message content, or a portion thereof, is described along with detailed information on its encoding, including hexadecimal constants and diagrams illustrating the structure. DRDA specifies the content and encoding for the portions of messages that it defines. Much of the remaining definition comes from DRDA-specific extensions to DDM and FD:OCA.

These fragmented definitions preclude software tools that automatically generate message encoders and decoders. The large number of constants and table look-ups required by a DRDA implementation must be either entered by hand, or possibly obtained from IBM through a special arrangement.

The number of sources for descriptions of message content and encoding requires the DRDA implementor to gradually become an expert in all the IBM architectures involved. Unfortunately, the relevant IBM architecture documents are the only printed source of information. The complexity involved is certain to require access to one or more *experts* on the subject. Today, the only DRDA experts are architects and key engineers at IBM.

Request Chaining and Asynchrony

SQL Access assumes that the underlying network provides full-duplex data transfer – and most networks do. The SQL Access protocol rules permit the client to send database requests without waiting for a previous database request to complete. By allowing *asynchronous* requests, the need for a request chaining mechanism is significantly reduced. This helps to simplify client design because there are very few rules that constrain the client's issuing of database requests.

The half-duplex data flow in LU6.2 significantly

constrains DRDA. The result of this is that DRDA includes description of *turn-to-send* processing in the actual architecture specification. DRDA introduces a set of rules for *chaining* a series of requests together. Using request chaining more than one request can be sent during a single turn to send. This helps to reduce the performance degradation associated often associated with half-duplex communication, but adds to complexity to the protocol and client design.

Protocol/Message Efficiency

The efficiency of a protocol is difficult to assess objectively without empirical measurements. In this section, some protocol characteristics that are believed to significantly affect performance are examined. These characteristics are: bandwidth utilization, repeated operations and asynchronous requests.

Bandwidth Utilization

Both SQL Access and DRDA use fairly bulky encoding schemes that include a significant amount of overhead. DRDA includes optimizations such as the re-use of metadata descriptors. The SQL Access BER compresses data values under certain, common circumstances.

Repeated Operations

Support for repeated operations is critically important for remote database access protocols. Efficient transfer of the rows of a table cannot be done at an acceptable rate using a protocol that cannot fetch or insert *n rows at-a-time*.

DRDA supports the concept of *block fetch* in which the unit of data retrieval is a *block* that may contain more than one data row. While this method is effective for retrieving data from a server, it does not address the case where a number of rows must be INSERTed into a table at the server. This *bulk* INSERT capability is not available in most SQL variants used by SQL application programmers, but it can and will be used by tools and utilities that issue database requests directly.

SQL Access defines a *repetition count* mechanism that permits any operation to be repeated one or more times. Each repetition may use a different set of input parameters, if desired. Through this mechanism, higher throughput data transfer can be achieved for both reads and writes.

Asynchronous Requests

Asynchronous requests are those that can be submitted to the server by the client without having to wait for some event such as the completion of a

previous request or a turn-to-send. Precompiled SQL applications cannot take advantage of asynchronous requests, but other types of applications, and the underlying client software, can achieve significant performance improvements when asynchronous requests are permitted. Only the database manipulation requests, that occur after connection set-up, are of interest.

A SQL Access client may issue asynchronous database requests to a server whenever it desires, provided that no more than 32 operations are outstanding at any time.

As discussed earlier, DRDA is constrained by the half-duplex nature of LU6.2. This means that a DRDA client may only send a database request to the server when all responses to previous requests have been received, and the client is granted the turn-to-send. This could result in significant delays for a client that received many asynchronous requests from some external stimuli.

Transaction Co-ordination

In their current state, both SQL Access and DRDA support only a single-phase, single-server transaction commit. Clients can invoke many servers, but the commitment of these servers is not co-ordinated. As a result, only a single client-server pair can be involved in an atomic transaction.

SQL Access, because it is based on RDA, will inherit sophisticated multi-site transaction co-ordination (two-phase commit) when the ISO Transaction Processing (TP) effort becomes a standard. RDA is already poised to exploit ISO TP.

DRDA will provide a definition for its approach to two-phase commitment in future *levels*. DRDA may take advantage of the two-phase commit protocol already supported by LU6.2.

Packages

DRDA introduces the concept of *packages*, persistently defined sets of one or more SQL statements stored at a DRDA server. Once a client has created a package at a particular server, it invokes individual statements one-at-a-time by identifying a package and a statement contained within. A client cannot execute any database language requests without using an existing package at the server. The creation of packages would typically be performed by a DBA, so a manual step is likely to be required before a client can access a particular server.

A DRDA server must support the creation, invocation and management of packages. The server must retain the definition of a package and

its contents. Servers may compile the SQL statements contained in a package at the time a package is created. In any event, a server must be able to tolerate SQL statements that it does not understand in packages that it manages, because the client application may create the same package at more than one type of server.

Packages are good for pre-planned, pre-compiled SQL in which a client accesses the same server in the same way, day after day. Packages are an inconvenience for ad hoc and decision support applications which dynamically attach to a server. With DRDA, a client *must* use packages to execute SQL statements – even if the client uses only dynamic SQL.

It is important to note that DRDA packages are not *stored SQL procedures*. The unit of invocation is a single SQL statement, and there are no flow control or error handling capabilities.

Packages or some form of persistent SQL will likely to be added to RDA, based on a proposal by IBM. Persistent SQL is a Phase II work item of SQL Access. ISO SQL has stored procedures as a work item. When these standards are formalized, they will become part of the SQL Access protocol.

Summary

SQL Access is based on international standards in which all companies can participate. It uses a common-subset approach – a single language, protocol, and encoding scheme is used universally. All servers speak and understand this Esperanto for SQL access to remote data. Many server-specific features can still be exploited using an escape clause, if desired. SQL Access is defined for the ISO standard network, OSI; but it can be redefined for any network that supports full-duplex communication sessions. Portability comes from the use of ISO standard SQL and X/Open portability guidelines.

DRDA is defined by IBM. It uses an *anything-goes* approach in which the client must be aware of the underlying data manager at each server. The *receiver-makes-it-right* encoding model optimizes interaction between similar platforms; but requires many different encoding formats be implemented when dissimilar platforms interact. It does not address portability beyond the IBM domain, and is dependent on SNA communication networks.

SQL Access and DRDA solve similar problems, but have different goals and orientation. Both approaches can be made to work in any situation. DRDA is optimized for a homogeneous network. SQL Access is optimized for heterogeneous

portability and interoperability.

References

- [1] *Structured Query Language (SQL)*, X/Open document, XO/PRELIM/91/030
- [2] *SQL Remote Database Access*, X/Open document, XO/SNAP/91/030
- [3] *ISO IS 9075:1989 - Database Language SQL* (equivalent to *ANSI X3.135-1989*; also known as SQL-89)
- [4] *ISO DIS 9075:199x – SQL2 Draft International Standard* (equivalent to *ANSI X3.135-199x*)
- [5] *ISO DIS 9579-1 - Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 1: Generic Model, Service, and Protocol and SQL Specialization*
- [6] *ISO DIS 9579-2 - Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 2: SQL Specialization*
- [7] *IBM Distributed Relational Database Architecture Reference* (SC26-4651)
- [8] *IBM Systems Application Architecture (SAA) Common Programming Interface Database Level 2, Reference* (SC26-4798)
- [9] *ISO IS 8824 - Information Technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*
- [10] *ISO IS 8825 - Information Technology - Open Systems Interconnection - Specification of Basic Encoding Rules (BER) for Abstract Syntax Notation One (ASN.1)*

Acknowledgements

The authors wish to thank the many individuals from database vendor and user companies for their valuable reviews of the material in this article. Special thanks to Richard Hackathorn for his numerous reviews and helpful comments.

Trademarks

IBM, Systems Application Architecture, SAA, DB2, SQL/DS, OS/2 EE, AS/400 and OS/400 are registered trademarks of International Business Machines Corporation.

Intel is a trademark of Intel Corporation.