Panel:
Ada and SQL

Moderator

Ben Brosgol
Alsys, Inc.

Panelists

Stephen Faris, *Oracle Corp.*
Marc H. Graham, *Software Engineering Institute*
James W. Moore, *IBM Federal Sector Division*
Jean-Pierre Rosen, *ADALOG*
S. Tucker Taft, *Intermetrics, Inc.*

Panel: Ada and SQL

Chair

Benjamin M. Brosgol
Alsys, Inc.
67 S. Bedford St.
Burlington, MA 01803
brosgol@ajpo.sei.cmu.edu
(617) 270-0030

Panelists

Stephen Faris
Oracle Corp.
500 Oracle Parkway
Redwood Shores, CA 94065
sfaris@oracle.com
(415) 506-6338

Marc H. Graham
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
marc@sei.cmu.edu
(412) 268-7784

James W. Moore
IBM Federal Sector Division
Mail Stop 182/3F10
800 N. Frederick Ave.
Gaithersburg, MD 20879
moorej@ajpo.sei.cmu.edu
(301) 240-7843

Jean-Pierre Rosen
ADALOG
27, avenue de Verdun
92170 Vanves
France
rosen@enst.enst.fr
+33 1 46 45 51 12

S. Tucker Taft
Intermetrics, Inc.
733 Concord Ave.
Cambridge, MA 02138
stt@inmet.inmet.com
(617) 270-0030

Panel Theme Statement

The usage of Ada with SQL is a pressing issue for the Information Systems community, but for various reasons the establishment of a widely-supported standard interface mechanism has been a tortuous process. Early efforts that attempted to fully exploit Ada suffered from implementation problems, and because of initial methodological opposition to the Embedded SQL approach, Ada has lagged behind other languages in supporting SQL.

The purpose of this panel is to survey where we are. Panelists will describe and critique the principal approaches (Embedded SQL, ANSI standard module language, SAME), describe current implementation and usage status/experience, and give recommendations on "where do we go from here." Ada 9X impact will be discussed.

Benjamin M. Brosgol

Benjamin Brosgol has been with Alsys, Inc. as Vice President / Technical Director since 1983. His involvement with Ada dates from the earliest days of the language effort. He was the leader of the Intermetrics "Red" language team during the Ada design competition in the late 1970's and has been an implementor, user, and teacher of Ada for over 10 years. Dr. Brosgol is the past-chair of ACM SIGAda and is the present chair of that organization's Commercial Ada Users Working Group (CAUWG). He is a Distinguished Reviewer for the Ada 9X Project.

Dr. Brosgol has a Ph.D. and M.S. in Applied Mathematics from Harvard University, and a B.A. in Mathematics, with honors, from Amherst College.

Position Paper from Jim Moore

The ANSI procedural or "module" binding from Ada to SQL is specified by citing two standards, X3.135-1989 and X3.168-1989 [A135, A168]. The 135 standard describes SQL in the form of a "module language". The module language permits the user to specify SQL functionality independently of any particular programming language. The SQL functions are represented as callable procedures within the module. The 168 standard specifies how Ada programs may make external calls upon the SQL procedures within the module.

The two standards permit another form of binding, the "embedded" binding. The embedded binding is semantically equivalent to the module binding; in fact, the embedded binding is defined in terms of the module binding. The difference occurs in the lexical appearance of the program. The embedded binding discards the separate SQL module and replaces each Ada procedure call upon the module with a stylized version of the text of the SQL statement which was to be called. The result is source code which intermixes SQL and Ada statements. The embedded binding is the traditional method for binding a programming language to SQL, but most Ada experts prefer the use of the module binding for reasons to be discussed later.

Before considering the relative advantages and disadvantages of the two forms of binding, an overview of the procedural binding will be given. An Ada program interfaces with an SQL module via an Ada package specification, often called the "concrete interface". The package spec and the SQL module share the same name and the SQL module contains a statement indicating that the interfacing language is Ada. Each SQL procedure in the module is matched by an Ada procedure specification in the concrete interface; the SQL procedure and the Ada procedure spec share the same name. Each SQL procedure has a number of formal parameters which are matched by identically named and ordered formal parameters in the Ada procedure spec. The parameter modes ("in", "out", etc.) are specified by the standard in a manner which is likely to agree with one's intuition. The permissible parameter types are declared in a package SQL_STANDARD which is specified by the standard. An example of an SQL module and the corresponding Ada concrete interface is shown in Figure 1. (For illustrative purposes, the figure adheres to the typographical convention of showing matching identifiers in lower case and everything else in upper case.)

The SQL standard provides a variety of rather primitive data types. Each language binding supports a subset of types which make sense for that language. In the case of Ada, the selected types are:

- CHAR (character strings)
- SMALLINT and INT (two possibly different sizes of integers)
- REAL and DOUBLE_PRECISION (two possibly different float sizes)

For the purposes of Ada, each of these types is declared in package SQL_STANDARD and decisions concerning range and precision are factored into that package. The package also declares an SQLCODE_TYPE for SQL return codes and a few other useful items. (The entire package is shown in Figure 2.) The standard forbids declaring other types in package SQL_STANDARD, so any vendor-specific types would have to be declared in another package supplied by the vendor.

There are some shortcomings in the ANSI SQL/Ada binding. First, the Ada binding has no decimal type because there is no decimal type for the Ada language itself. Second, the level of the interface is somewhat primitive- -it uses numeric return codes rather than exceptions and provides little support for user-defined types. Third, the binding does not ensure safe treatment of database "nulls"--the application programmer must be aware of the possibility and code accordingly.

The latter two shortcomings motivated much of the work of the SQL Ada Module Extension Design Committee (SAME-DC) at the Software Engineering Institute. Beside recommending the binding that was eventually adopted by ANSI, the committee also developed the "SAME method" [GRA, MOO1], a set of guidelines for creating an "abstract interface" which corresponds to the abstract interface but which is at a higher level of abstraction.

One of the disadvantages of using the ANSI procedural binding and the SAME method in a manual manner is that one ends up writing essentially equivalent information in several different places, ie in the SQL module, in the Ada concrete interface, and in the Ada abstract interface. This has led some developers, eg. Susan Phillips [LECL], to develop tools which may be used to generate the various interfaces from a single statement. The most notable effort is headed by Marc Graham [CHA] to provide an SQL/Ada Module Description Language (SAMeDL) which when processed by appropriate tooling would produce the required modules and interfaces.

There are some relative advantages and disadvantages of the several methods:

The Procedural Module Binding vs. the Embedded Binding

- The procedural binding keeps Ada code and SQL code in separate modules. Tools which process Ada source code won't be "broken" by the appearance of SQL statements.

- The procedural binding enforces modularization of database accesses into a single place.

The Procedural Module Binding vs. the SAME Approach or SAMeDL

- The procedural module binding is standardized already and is being provided by vendors. Standardization of the SAMeDL is likely to take a few years.

- Using the SAMeDL requires using a third language and a third set of tools.

- The SAME approach enforces a view where each call to the database at the abstract level is implemented by a single SQL statement. This may not be the desired abstraction. For example, if one is implementing a double-entry bookkeeping system, one might want a single abstract transaction to generate a journal operation and two postings to the database.

Figure 1. Example of Module Binding

SQL Module:

```
MODULE example_module
LANGUAGE ADA
...
PROCEDURE get_amt_in_stock
      :pn          INT
      :quan        INT
      sqlcode;
   SELECT DISTINCT P.QUANTITY INTO :QUAN FROM P
      WHERE P.PARTNO = :PN;
...
```

Ada Concrete Interface (package specification):

```
WITH SQL_STANDARD;
USE  SQL_STANDARD;
PACKAGE example_module IS
      ...
      PROCEDURE get_amt_in_stock
            ( pn          : IN  INT ;
              quan        : OUT INT ;
              sqlcode     : OUT SQLCODE_TYPE );
      ...
END example_module;
```

Note that lower case letters are used to mark identifier names which match between the SQL module and the Ada package specification.

Figure 2. Package SQL_STANDARD

```
package SQL_STANDARD is
   package CHARACTER_SET    renames csp;
   subtype CHARACTER_TYPE   is CHARACTER_SET.cst;
   type    CHAR             is array (POSITIVE range <>) of
                               CHARACTER_TYPE;
   type    SMALLINT         is range bs..ts;
   type    INT              is range bi..ti;
   type    REAL             is digits dr;
   type    DOUBLE_PRECISION is digits dd;
   type    SQLCODE_TYPE     is range bsc..tsc;
   subtype SQL_ERROR        is SQLCODE_TYPE range
                               SQLCODE_TYPE'FIRST..-1;
   subtype NOT_FOUND        is SQLCODE_TYPE range 100..100;
   subtype INDICATOR_TYPE   is t;
end SQL_STANDARD;
```

References

[A135] American National Standard for Information Systems-Database Language-SQL, American National Standards Institute, X3.135-1986.

It was superseded by American National Standard for Information Systems-Database Language-SQL with Integrity Enhancement, American National Standards Institute, X3.135-1989.

[A168] American National Standard for Information Systems-Database Language-Embedded SQL, American National Standards Institute, X3.168-1989.

[CHA] G. Chastek, M. Graham & G. Zelesnik, The SQL/Ada Module Description Language - SAMeDL, Software Engineering Institute, SEI-90-TR-26, 1990.

[GRA] Marc H. Graham, Guidelines for the Use of the SAME, Software Engineering Institute, CMU/SEI-89-TR-16, 1989.

[LECL] Allison LeClair and Susan Phillips, "A Prototype Implementation of the SQL-Ada Module Extension Method", ACM Tri-Ada'90 Conference, Baltimore, MD, December 1990.

[MOO1] James W. Moore, Conformance Criteria for the SAME Approach to Binding Ada Programs to SQL, Software Engineering Institute, SEI-89-SR-14, 1989.

[MOO2] James W. Moore, "The ANSI Binding of SQL to Ada", ACM SIGAda AdaLetters, Vol. XI, Number 5 (July/August 1991).

James W. Moore

Mr. Moore, with IBM since 1969, is one of those responsible for introducing IBM to Ada, on the SubACS program several years ago. His current interests involve the creation of secondary standards facilitating the

creation and reuse of Ada programs, notably SQL and POSIX. He is the technical lead for DARPA's ASSET Reuse Library program and is a member of the DoD's Ada Federal Advisory Board.

Mr. Moore has an MS from Syracuse University and a BS from the University of North Carolina.

Position Paper from Jean-Pierre Rosen

There are two different views of the need for a binding of Ada to SQL. On one side, people developing Ada programs may wish to access data that are stored in a data base. To those people, a data base is simply a sophisticated IO system they want to access using normal Ada usage: a set of standardized packages.

On the other side, people who are used to SQL want to use Ada for its expressive power; embedding SQL into Ada allows them to develop data base applications with the algorithmic power of Ada.

Is it possible to provide a single interface that will satisfy both "Ada first" and "data base first" users? Unfortunately not, because the two views share different models of the notion of typing. In Ada, types belong to a program. There is no "basic type": even integer types are user defined, and all types are unique to the program that declares them. But all types are static: it is possible by inspecting the code to know the full set of types that are used by the application, as well as the
characteristics of all of them.

In SQL, there is a small set of basic types: INTEGER, CHARS, DATES... All structures (rows, tables) are a combination of these basic types. But these combinations are fully dynamic: only by examining a request together with the structure of the data base at the time the request is executed can you determine the structure of the result.

It is therefore impossible to provide a binding that will preserve both typing models: the variety of types in Ada disappears when values are stored in the data base, and the dynamic model of SQL must be frozen to match Ada static types. Tradeoffs must be chosen between those opposite views, and the "best tradeoff" may vary according to the kind of usage of the interface.

For these reasons, we believe that two levels of interface between Ada and SQL are necessary: a low level interface and a high level one. The low level interface should be a "pure Ada" interface. Actually, it should be a set of standard packages allowing the Ada programmer to access data base operations in a standardized way. Being fully standard Ada, it is not possible to make any verification with the data base types at compile time. The kind of possible type checking would be no more (but no less) than what is provided by the usual IO packages. Such an interface would be fully compliant with the requirements for binding Ada to SQL, as set forth by the SQL rapporteur group of ISO/WG9. We have proposed the scheme for such an interface, and a prototype implementation has been developped recently to demonstrate feasibility.

The high level interface should take into account the structure of the data base and provide extended type checking. The SAME addresses this level of interface by providing a new language from which SQL and Ada modules can be derived. Compilation of the SAMEDL inspects the data base structure to provide compile time type verification.

Note that the low level interface should be designed in order to provide a convenient mean for implementing the high level one.

The issue of providing access to a data base from an Ada program is a complex one; only by recognizing that there are different levels of abstractions in the needs for interfaces, and that several bindings should be provided for each of these levels, can a satisfactory solution be found for the different kinds of users.

Jean-Pierre Rosen

Dr. Rosen is the founder of Adalog, a company specialized in high level training and consultancy in the fields of Ada and OOD. Previously he was a Professor at the Ecole Nationale Superieure des Telecommunications (ENST), where he taught Software Engineering and Ada. Dr. Rosen is Chairman of Ada France, member of the board of Ada Europe, and member of AFNOR and ISO groups on Ada.

Dr. Rosen received a Ph.D. from ENST.

Position Paper from Tucker Taft

The most difficult language problem in the interface between SQL and Ada is in communicating ad hoc queries defined in terms of expressions involving both Ada variables and SQL tables.

The SQL Module and SAME approaches avoid this problem by defining a procedural interface between the Ada and SQL "worlds." These approaches have other benefits, in that they encourage a clear separation between database-dependent parts of a program and other parts. In addition, they allow a database-specific processor to perform static analysis of the queries prior to run-time, allowing for the possibility of greater query optimization.

On the other hand, the modular approaches can be seen as creating an arbitrary split between the persistent and non-persistent worlds. Object-oriented database systems have tended toward trying to minimize this split, by more tightly integrating the programming language and the persistent storage facilities.

Now that it appears very likely that Ada 9X will provide more direct support for object-oriented programming techniques, it is worth reexamining the best way for Ada programs to tap into the power and flexibility of relational database systems. It may be that a more "object-oriented" interface can be defined, bringing the concepts of persistent data tables into the language as a more nearly first class abstraction.

Nevertheless, it will still be essential that there be a robust and standard interface between Ada and "vanilla" SQL, and we continue to believe that the SAMe-DL provides the best current combination of portability, flexibility, safety, usability, and efficiency.

S. Tucker Taft

Mr. Taft has worked for Intermetrics Inc. since 1980 and now holds the position of Technical Director of the Ada Division. He is currently the Lead Engineer on the Ada 9X Mapping/Revision Team. Prior to 1990, Mr. Taft was responsible for technical integration of the Ada Integrated Environment, including the compiler and other support tools. He has participated in the SQL Ada Module Extensions Design Committee (SAME-DC) chaired by Dr. Marc Graham of the SEI, and was also involved in the Ada-83, CAIS, and Posix/Ada standardization efforts.

Mr. Taft graduated Summa Cum Laude from Harvard College with an A.B. degree in Chemistry.

Stephen Faris

Stephen Faris has been with Oracle Corp. since 1988, and he is currently a Senior Product Manager in the Systems Division. His responsibilities encompass the Product Management activities for the Languages Products group, whose charter is to develop Call-level, Embedded SQL, and Module Language interfaces to the ORACLE RDBMS and tools. The group is responsible for Oracle's Procedural Language Extension to SQL

(PL/SQL) as well, which interestingly is based upon a smooth integration of Ada and SQL syntax and features.

Mr. Faris holds an MS degree in Computer Science from UCLA and a BS in Engineering from the University of Michigan.

Marc H. Graham

Marc Graham is a Senior Computer Scientist at the Software Engineering Institute where he has been an active participant in the development of an Ada/SQL interface. Prior to joining the SEI in 1987, Dr. Graham was Director of the Software Corporate Technology Center for Sperry, an Associate Professor of Computer Science at Georgia Tech, and developer of Cullinane Corporation's database retrieval software.

Dr. Graham received a Ph.D. in Computer Science at the University of Toronto and a Bachelor's degree at Wesleyan University. He has published a number of papers on the theoretical aspects of database semantics and transaction processing.