



USAISEC

AD-A268 517

②



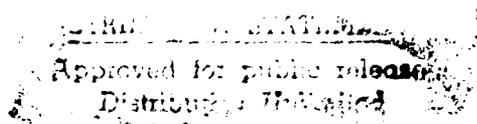
*US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300*

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

**SAMeDL:
What Is It?
Why Is It Important?
Who Can Help?**

ASQB-GI-92-014

September 1992



**AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800**



93-19668



93 8 24 004

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

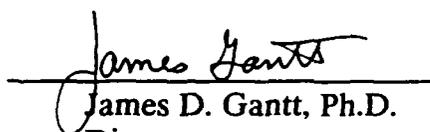
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS NONE	
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT N/A	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A			
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A	
6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION N/A	
6c. ADDRESS (City, State, and Zip Code)		7b. ADDRESS (City, State, and ZIP Code) N/A	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Software Technology Branch, ARL	8b. OFFICE SYMBOL (If applicable) AMSRL-CI-CD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800		10. SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) SAMeDL: What Is It? Why Is It Important? Who Can Help?			
12. PERSONAL AUTHOR(S) MS. Deb Waterman			
13a. TYPE OF REPORT Technical Paper	13b. TIME COVERED FROM Apr 91 TO Sept 92	14. DATE OF REPORT (Year, Month, Day) Sept 15, 1992	15. PAGE COUNT 15
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	Ada Database Access, SAMeDL, Ada extension module, SQL	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
<p>This report details the research efforts into the SQL Ada Module Database Description Language (SAMeDL). Four compilers are presented (Oracle, Informix, XDB, and Sybase) that allow Ada application programs to access database using a standard SQL query language. Copies of the compiler can be obtained from the DoD Ada Joint Program Office 703/614-0209.</p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL LTC David S. Stevens		22b. TELEPHONE (Include Area Code) (404) 894-3110	22c. OFFICE SYMBOL AMSRL-CI-CD

This research was performed by Statistica Inc., contract number DAKF11-91-C-0035, for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U. S. Army Information Systems Engineering Command (USAISEC). This final report discusses a set of SAMeDL compilers and work environment that were developed during the contract. Request for copies of the compiler can be obtained from the DoD Ada Joint Program Office, 703/614/0209. This research report is not construed as an official Army or DoD Position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED



Glenn E. Racine, Chief
Computer and Information
Systems Division



James D. Gantt, Ph.D.
Director
AIRMICS

DTIC QUALITY INSPECTED 3

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

SAMeDL

What is it?
Why is it Important?
Who Can Help?

SAMeDL: What is it? Why is it important? Who can help?

Background

As the Ada programming language becomes more widely used for information systems development, a viable interface between Ada and off-the-shelf database management systems (DBMS) is a critical requirement for success. To ensure longevity of the applications software, as well as to enhance maintenance and reliability, most Ada information systems developers prefer to use established standards to support their development activities. For information systems, this preference for standards means using Structured Query Language (SQL) as a means of accessing commercial DBMS products.

SQL and Ada are both ANSI (American National Standards Institute) and DoD (Department of Defense) standards. Both standards are also required for the development of DoD information systems, and for other applications which require the use of a DBMS (i.e., command and control systems). Unfortunately, the task of interfacing Ada and SQL is complex and non-trivial in scope. The underlying computational models for the two languages are very different, and they were not explicitly designed to work with each other.

Although SQL is an ANSI standard, most DBMS implementations of the standard offer "extensions" which provide more power to DBMS users. These non-standard features provide pragmatic real-world capabilities to users, while at the same time significantly complicating the task of information systems developers. If a developer makes extensive use of non-standard SQL features, the portability of the application can be severely degraded.

In 1989, ANSI promulgated two Ada/SQL bindings that specify standard approaches for binding Ada applications to SQL databases. Unfortunately, the ANSI bindings do not support user-defined types and the safe handling of "null" values in a database. Furthermore, the ANSI standards do not enforce the extensive compile-time checks that Ada developers have come to rely on, requiring instead the use of expensive run-time error traps.

Impact on Developers

The benefits normally associated with the use of Ada (i.e., maintainability, reusability, portability, etc.) are primarily dependent on the application of sound software engineering principles. Use of proven software engineering concepts, such as information hiding, abstraction, modularity, etc., can pay off in the form of lower maintenance costs, and higher levels of portability and reuse. Ada readily supports the application of sound software engineering techniques and methods, which is the dominant strength of the language.

By comparison, SQL was developed to support the definition, manipulation, and control of data in a relational database. As a special-purpose language for database operations, SQL was not designed to be used in conjunction with a powerful programming language such as Ada. Unlike Ada, SQL does not provide support for applying the principles of software engineering. The requirement for using SQL in combination with Ada applications code can cause major problems in accuracy, performance, portability, and reuse.

Embedded Versus Module SQL Approach

Historically, vendors of SQL products almost always provide access to a DBMS by allowing users to "embed" SQL statements in their application software. In the "embedded" approach, the developer places DBMS manipulation (SQL) statements at the appropriate points in the application code, intermixed with the code written in the application's programming language. A pre-processor then takes out the SQL instructions and replaces them with the DBMS-specific calls.

A "module" approach, by comparison, encapsulates all of the SQL statements in separate modules. The application program then makes calls to the SQL modules, rather than having the SQL statements intermixed with applications code throughout the software system. For the ANSI Ada/SQL bindings, both an embedded and a procedural or module form are included. In the procedural form, access to the DBMS is through Ada procedure calls to the SQL procedures contained in the SQL module.

The embedded SQL approach results in applications code that contains SQL statements throughout the entire program. Given the propensity of DBMS vendors to provide "extended" SQL features in their products, the result of an embedded SQL approach is an application that is inextricably tied to a particular DBMS. An information system developer sacrifices virtually all portability and future reuse when using an embedded SQL approach.

In the module approach, the ability to encapsulate the SQL statements in a single module provides an opportunity for increased portability and independence from product-specific features. The module approach is more in line with the application of sound software engineering principles, and it allows the information systems developer to more thoroughly exploit the power and features of the Ada language.

Although the ANSI Ada/SQL bindings include both an embedded standard and a module standard, vendors are not required to provide both forms of the bindings to be ANSI compliant. Most DBMS vendors offer an embedded Ada/SQL capability, and the embedded form is expected to be the predominant product offering in the future. Even though the module form of the ANSI binding helps in developing more portable applications, it does not support user-defined types and the safe handling of "null" values in a DBMS.

The Importance of Handling "Null" Values

The safe handling of "null" values in a database is of special concern to information systems developers. Null values are peculiar to relational database processing, as they indicate missing or unknown information. The Ada language itself does not have a means of identifying or handling unknown data. By comparison, SQL provides a set of operators which handle incomplete information. These operators are used by SQL for its own processing, but the operators are not "exported" to the application programming language.

In SQL, data objects in an application program have two parts: a variable which holds the value of the item; and an "indicator" which indicates whether or not the value is null. In other words, the "indicator" shows whether the value is meaningful for the operation being performed. Improper handling of these indicators can lead to subtle software errors, where seemingly valid results are wrong.

A good real world example of the impact of a null value would be the instance in which a database concerning personnel information is polled to calculate the average age of employees. Since an employee is not required to disclose his/her age, the age field in the database may be missing, or "null." In this example, if the null

value is included in the calculation (no value for the age, but the employee is included in the divisor for the average), the result is not accurate.

SAMeDL—Its origins, implementation, and benefits

The challenges pertaining to the use of Ada and SQL in a well-engineered information system led to the formation of the SQL Ada Module Extension Design Committee (SAME-DC) at the Software Engineering Institute (SEI). The committee developed the SQL Ada Module Extension (SAME) methodology to support and extend the modular approach to using Ada and SQL in database applications. The language which is used to implement SAME is the SQL Ada Module Description Language, or SAMeDL.

Using the SAME methodology requires that SQL statements be separated from the Ada application code, and encapsulated in separate modules. The SQL statements are not embedded in the Ada packages, thus isolating the Ada application from the DBMS design and implementation.

SAMeDL is designed to facilitate the construction of Ada database applications that use the SAME methodology. The SAME method involves the use of an abstract interface, an abstract module, a concrete interface, and a concrete module.

- The abstract interface is a set of Ada package specifications containing the type and procedure declarations to be used by the Ada application program.
- The abstract module is a set of bodies for the abstract interface. These bodies are responsible for invoking the routines of the concrete interface, and converting between the Ada and the SQL data and error representations.

- The concrete interface is a set of Ada specifications that define the SQL procedures needed by the abstract module.
- The concrete module is a set of SQL procedures that implement the concrete interface.

To implement the SAME methodology, SAMeDL provides the language constructs from which three types of modules are written: the definition module; the schema module; and the abstract module. The three modules together comprise a compilation unit for the SAMeDL compiler. The definition module contains the declarations of domains, constants, records, enumerations, exceptions, and status maps. The schema module contains the declaration of tables, views, and privileges. The abstract module contains procedure and cursor declarations.

The SAME methodology uses Ada features to provide the following services to Ada/SQL applications:

- Safe treatment of SQL data that may contain null values. SAMeDL provides robust treatment of SQL data within application programs which effectively prevents use of null values as though they were not null. The SAMeDL treatment requires no run-time conversion of non-null data.
- Flexible handling of DBMS errors and exceptional conditions. This feature allows application designers to decide which conditions are expected and which are irrecoverable. However, it prevents any such DBMS conditions from being "missed" by the application.
- Enforcement of a strong typing discipline to SQL statements and user-defined types. This service provides

an extended database description using abstract, application-oriented types as well as the strong typing discipline for the SQL statements.

Applying the SAME methodology enhances the following attributes of an Ada information system:

- **Conceptual Clarity** It is good design practice to separate the database statements in an application from the application logic. This allows the application's information needs to be captured during the design.
- **Portability** The great bulk of the code of any database application lies in the application's logic, as distinct from the database logic. Since SAME Ada applications contain no SQL statements, they port from one DBMS to another without modification.
- **Simplicity** Given the differences between Ada and SQL (i.e., the syntax and semantics of the two languages), coding in both languages simultaneously is very confusing.

These attributes have significant payoffs for the developer. The SAME methodology, by virtue of its enforced modularity and encapsulation, supports the use of object-oriented software development for Ada-based information systems. The conceptual clarity of the approach contributes to the early detection of errors in the development cycle, which helps to lower the cost of finding and fixing software problems during development.

The modular, encapsulated nature of SAME also facilitates the management of the development team, and aids in the assignment of discrete tasks to staff members with particular levels of expertise. For example, SQL experts can focus on the statements required to operate the DBMS being used, while the Ada software engineers can

concentrate on the design and development of the Ada application code. The simplicity of working in a single language pays off significantly in the form of increased productivity, higher quality, fewer errors, more straightforward project tracking, and higher team morale.

Real World Experience with SAMeDL

In September 1991, Statistica, Inc., in Reston, Virginia, was awarded an Ada Technology Insertion Program (ATIP) contract to explore the merits of SAME on a real information system application. The project used SAMeDL to redesign the SQL interface of an existing application, in this case the U.S. Army's Standard Installation Division Personnel System (SIDPERS-3). SIDPERS-3 was originally designed using another Ada/SQL binding.

Statistica approached the project in two phases. The first phase involved the replacement of the SIDPERS-3 SQL binding layer with a duplicate layer implemented in SAMeDL. The Ada application design and code remained unchanged. The purpose of the first phase was to determine the ease of converting an existing Ada/SQL application to SAMeDL. In the second phase, Statistica redesigned and re-implemented the SQL binding, to demonstrate the benefits of an up-front SAME design approach.

Phase 1 Approach

In the first phase of the project, Statistica identified two Computer Software Units (CSUs) from the SIDPERS-3 application for re-implementation using SAMeDL. These CSUs, listed in Table 1, were selected on the basis of the functionality and services required of the database. To facilitate the analysis and measurement of performance, each CSU in the application layer represents a complete thread of control.

CSU Title	Description	Database Services	Approx. SLOC*
Sgt Seg Promotion Eligibility Unit	This CSU initially calculates a soldier's promotion points and generates PCN AAA-209, DA Form 3355-E, Promotion Point Worksheet	Select Update Insert Fetch	2,866
Promotion Standing List Removal Action	This CSU removes a soldier from the E5-E6 Promotion Standing List and generates PCN AAA-034, a Removal From Local Recommended List memorandum	Select Delete	2,150

* Source Lines of Code

Table 1: CSUs in the Ada Application Layer

The original SIDPERS-3 implementation used a commercial database product, XDB, which includes an SQL module compiler that provides the binding between the Ada application code and the XDB relational database product. Using the XDB binding, the programmer declares the SQL types and statements required by the application in the SQL Module. As shown in Figure 1, the SQL Module Compiler processes the SQL Module to generate Ada package specifications and bodies. As shown in Figure 1, the SQL Module Compiler processes the SQL Module to generate Ada package specifications and bodies.

With the XDB binding, the Ada packages are dependent upon the SQL_Standard, SQL_Definitions, and Dynamic_SQL packages supplied by XDB. At this point in the process, the SQL interface is compiled with the application coded and linked into the Ada executable.

Figure 2 (next page) shows the design strategy used by the SIDPERS-3 development team. Early design decisions were predicated on the use of XDB database product, with its product-specific Ada/SQL binding. Since neither the binding nor the database product itself support strong typing, the SIDPERS-3 design team created a layer of Type

packages. These packages declare subtypes which correspond to each column in the database.

The SIDPERS-3 team also developed a Man-Machine Interface (MMI) to handle all user interfaces, including reports. Data is passed between the MMI and the Ada application code as string objects. The Types packages mentioned in the previous paragraph also serve the MMI by eliminating a conversion layer between the application and the MMI.

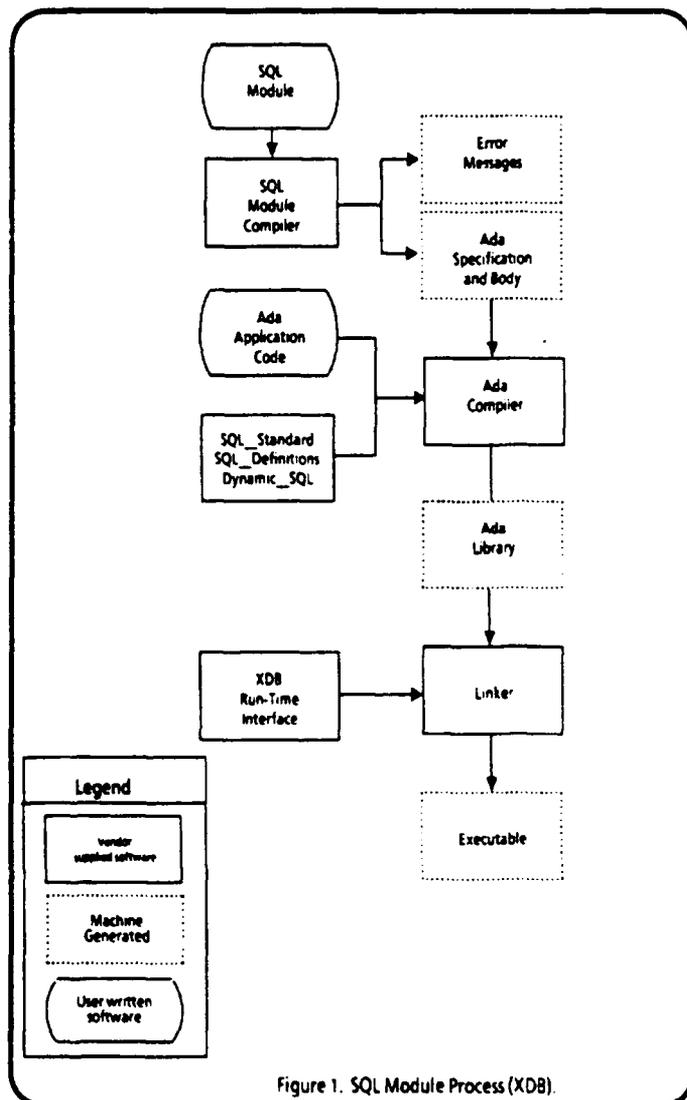


Figure 1. SQL Module Process (XDB).

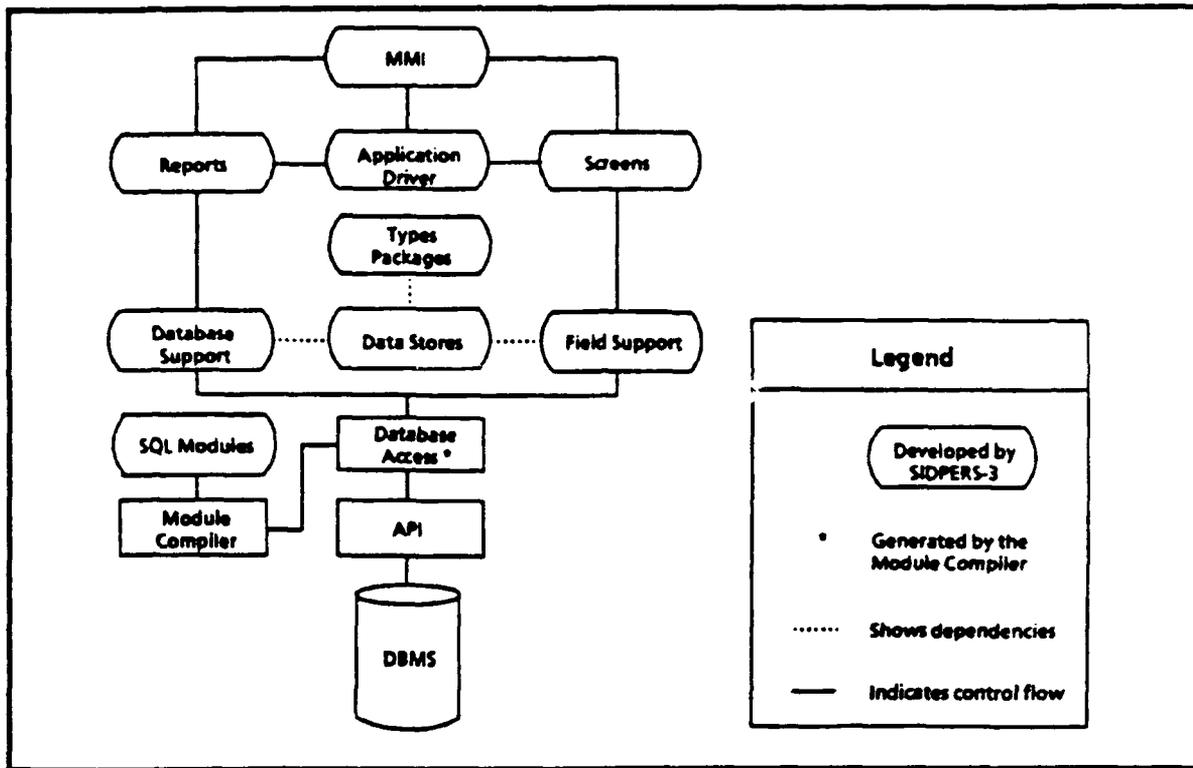


Figure 2. The SIDPERS-3 Prototype Design Strategy

To avoid the bulky processing required to test the “nullness” of data values, the SIDPERS-3 design team created the database with NOT NULL columns. In the Database Support Layer, data retrieved from the database is explicitly converted to SIDPERS-3 types before processing by the application layer. The Database Support layer isolates the SIDPERS-3 application code from changes that may occur to the database.

In the first phase of the SAMeDL project, the SQL Module files (written in SQL module language) were replaced with SAMeDL modules. This step was completed quickly, since only certain tables and columns were required for the selected CSUs. The SAMeDL Modules were then submitted to the the SAMeDL compiler for generation of the Ada packages. The generated Ada packages replaced the Database Access files from the original SIDPERS-3 model (refer to Figure 2).

The next step in the first phase of the project was the modification of the Database Support packages. Minor changes were made to these packages, to remove Commit_Work and Rollback procedures. Since those procedures are provided as standard constructs by SAMeDL, they were declared in the SAMeDL abstract modules. Procedures or functions were added to the Database Support packages to convert the SAMeDL types to SIDPERS-3 data types, to minimize changes to the application layer. Finally, the exception handlers were removed and replaced by the SAMeDL error handling routines.

First Phase Results and Analysis

The cost of converting to SAMeDL on the SIDPERS-3 project was low. The ease of conversion can be attributed to the Module binding provided by XDB and the layering approach used to isolate the database from the applications code. With the exception of some error handling, the

conversion did not derive any benefits from converting to SAME. The strong typing was lost when the data moved into the application layer. Also, since the database was created with NOT NULL columns, the application did not take advantage of SAMeDL's handling of null fields.

The single SAME-related benefit on the SIDPERS-3 conversion was a further isolation of the application software from a change to another DBMS. That benefit would be negated, however, if the new DBMS did not support the SAME architecture.

Several issues should be addressed when considering the adoption of SAME for Ada database applications. This is especially true for an application where the preliminary design has already been completed. Some of the more important issues include the following:

- How is the MMI designed? Can the MMI be easily altered to use SAMeDL types? The MMI developed for SIDPERS-3 was designed to meet the specific requirements of SIDPERS-3. The MMI, therefore, directly controlled the design of the application. The original SIDPERS-3 implementation was designed with weaker user-defined types throughout the application layer to meet the interface requirements of the MMI. Converting to SAMeDL would require major modifications to the application layer.
- What is the current database interface? Does the current design isolate the database? Does the application directly reference the SQL types?
- What is the size of the system? The number of changes and level of effort for a SAMeDL conversion increase proportionately with the size of the system.

The "bottom line" result of the first phase effort was that no value was added by converting

the SIDPERS-3 SQL interface to the SAME architecture. To realize the extensive benefits of SAME, the SIDPERS-3 application would have to be redesigned and re-implemented using SAMeDL—an effort which was the impetus for the second phase of the project.

Phase 2—Redesign and Re-implement with SAMeDL

The second phase of the SAMeDL project involved the redesign and re-implementation of the same SIDPERS-3 CSUs that were converted in the first phase. Instead of a simple conversion, and rather than starting with a new application, the project team redesigned and re-implemented the same CSUs that were listed in Table 1. This approach was taken to ensure an accurate baseline for comparisons between the design methodologies, with both methods being used on the same functional application.

The second phase was implemented using five steps:

1. **Analyze the data requirements of the CSU to identify objects or specific data groups.** For example, data identifying a soldier, such as social security number (SSN) and name, comprise a Soldier object. The data identifying a Unit (i.e., Unit Identification Code (UIC) and name) are attributes of the Unit object.
2. **Build SAMeDL domains and support packages around each object.** For each object, a SAMeDL definition module was written to declare the attribute domain of the object. A corresponding SAMeDL abstract module was written to provide all of the database operations required to use the object.
3. **Modify the database schema to represent the real world.** Where appropriate, the project team changed

SAMeDL: What It Is, Why It's Important, Who Can Help

database columns to allow null values. The project team then created a SAMeDL schema module to match the new database schema.

4. **Compile SAMeDL modules.** For each definition module, the SAMeDL compiler generates an Ada specification package in which derived types are declared for each domain. Additionally, the SAMeDL compiler creates a set of Ada packages for each abstract module. This set of Ada packages is the abstract interface defined in the SAME architecture. In the XDB version of the SAMeDL compiler, a third set of packages (actually generated by the XDB module compiler) is required to

implement the database calls in the abstract module. This third set of packages becomes the concrete interface in the SAME architecture.

5. **Redesign and modify the application layer.** The project team identified several goals in redesigning the application layer:

- a. Replace the weaker subtypes in the Types package with SAMeDL derived types. The objective of this goal was to remove the redundant type declarations and enforce compile-time checks through derived limited private types.

- b. Hide implementation details of the Data Stores by encapsulating Data Stores within the support packages.

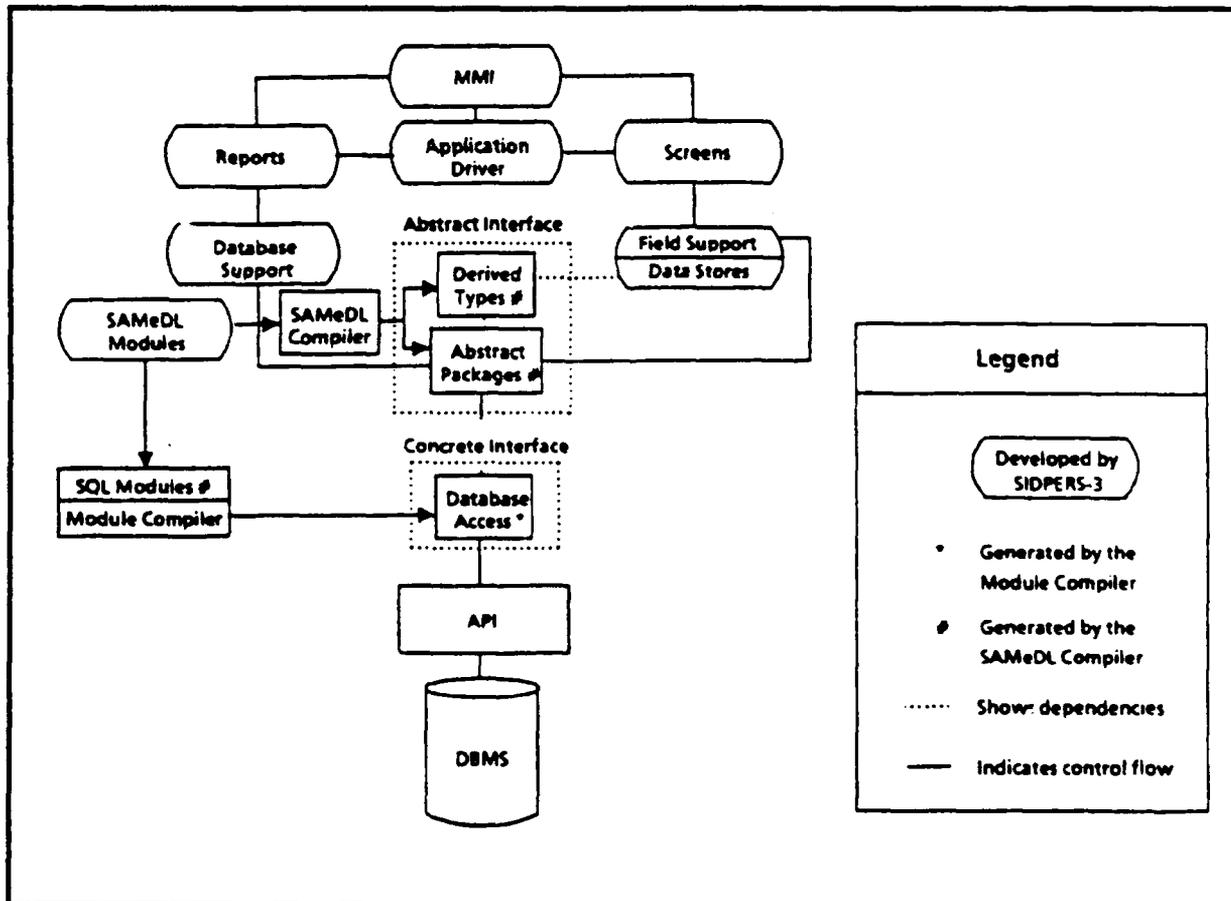


Figure 3. The SAMeDL Redesign Architecture

The objective for this step was to utilize information hiding and to give the design a more "object-oriented" flavor.

c. Retain strong data typing up to the point where the MMI controls the data. The objective here was to process the data using the operations defined for each data type and take advantage of compile time, rather than run time, error detection.

The resulting SAMeDL Redesign Architecture is shown in Figure 3.

Phase 2 Analysis

To take full advantage of the many benefits offered by the SAME architecture, SAMeDL must be the basis for the design of an Ada/SQL database application. Once the database design has become stable, the SAMeDL modules can be designed and written. Database stability is the key to success in implementing SAMeDL—since the application layer is built on top of the SAMeDL data types, any change in the database schema resulting in a change to the SAMeDL types

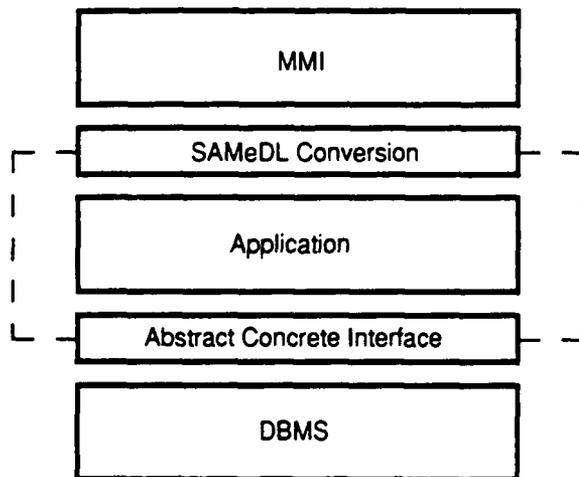


Figure 4: The SAMeDL Conversion Layers

requires modifications to, and recompilation of, the application layer.

To maintain strong typing, either a conversion layer must be inserted between the application and the MMI (see Figure 4), or the MMI must be designed to reference the SAMeDL types directly.

Conclusions

SAMeDL is another programming language that can be as complex to use as is Ada. It is a hybrid of both Ada and SQL, offering the best features of Ada, and allowing the user to specify database services in SQL-like statements. The user must therefore be proficient in both Ada and SQL, while learning a third programming medium. Program managers must consider the impact of training and learning curve delays on SAMeDL application development schedules.

The SAME methodology provides a mechanism for enforcing good software engineering during the development of new information systems. However, the gains in quality are not free. Database stability, user interface design, and user training are factors that must be considered by any project developing a new application with SAMeDL.

SAMeDL Tools

As a part of the SAMeDL project on the SIDPERS-3 application, a SAMeDL toolset was developed for four commercial database management products (Informix, Oracle, Sybase, and XDB). Intermetrics, Inc., developed the SAMeDL tools that were used during the project.

The toolset includes a SAMeDL compiler and Module Manager. The Module Manager is a library manager that maintains source code control and performs consistency checks on SAMeDL source code and its corresponding Ada/SQL interface.

**This document is a publication of the U.S. Government,
produced for the Ada Joint Program Office.
It is approved for public release.
Distribution unlimited.**

Points of Contact for More Information

**Ada Joint Program Office (AJPO)
Room 3E118
The Pentagon
Washington, DC 20301-3081
703-614-0208**

**Ada Information Clearinghouse (AdalC)
c/o IIT Research Institute
4600 Forbes Boulevard
Lanham, MD 20706-4320
800-232-4211 or 703-685-1477**

**Statistica, Inc.
Ms. Deb Waterman
12200 Sunrise Valley Drive
Suite 300
Reston, VA 22091
703-758-2533
FAX: 703-758-0641**

**Intermetrics, Inc.
Gary Tominovich
7918 Jones Branch Drive
Suite 710
McLean, VA 22102
703-827-2606**

AD NUMBER	DATE	DTIC ACCESSION NOTICE
1. REPORT IDENTIFYING INFORMATION		
A. ORIGINATING AGENCY <i>US AIRMAILS 1522</i>		REQUESTER: 1. Put your mailing address on reverse of form. 2. Complete items 1 and 2. 3. Attach form to reports mailed to DTIC. 4. Use unclassified information only. DTIC: 1. Assign AD Number. 2. Return to requester.
B. REPORT TITLE AND/OR NUMBER <i>SAME DL what is it</i>		
C. MONITOR REPORT NUMBER <i>HSQB-CJ-97-C14</i>		
D. PREPARED UNDER CONTRACT NUMBER <i>N/A</i>		
2. DISTRIBUTION STATEMENT <i>unlimited</i>		

DTIC FORM 50
DEC 80

PREVIOUS EDITIONS ARE OBSOLETE