DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/91D-04

ANALYSIS OF A DECISION SUPPORT SYSTEM FOR CASE TOOL SELECTION AND THE SPECIFICATION OF AN ADA TO SQL ABSTRACT INTERFACE

THESIS

Tina M. DeAngelis
Captain, USAF

AFIT/GCS/ENG/91D

AFIT/GCS/ENG/91D-04

ANALYSIS OF A DECISION SUPPORT SYSTEM FOR CASE TOOL SELECTION AND THE SPECIFICATION OF AN ADA TO SQL ABSTRACT INTERFACE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for the Degree of Master of Science (Computer Science)

Tina M. DeAngelis, B.S.E.E.

Captain,USAF

September 1991

Acknowledgments

I would like to thank my advisor Lt Col Lawlis whose patience and guidance and previous contributions to the field of Decision support tools helped to get this research accomplished. I would also like to thank Major Roth whose expert assistance in the database arena, helped me to accomplish a data analysis of the STEMdB tool. I would next like to acknowledge and thank the people from Draper Labs, Jim Van Buren and Rick Hildebrant for providing me with documents, prototypes and answers pertaining to the STEMdB tool. I would next like to thank Gary Petersen of the STSC whose professionalism helped to insure that I received the information I needed to get the research done. I would like to thank my readers Maj Howatt and Maj Gunsch for being on my committee and for providing me with great feedback. I would also like to thank my mother and sister whose continued encouragement kept me going. Finally, I would like to praise God for helping to get me through this thesis effort.

Table of Contents

Page

List of Figures

List of Tables

List of Acronyms

ACRONYM	NAME
ASSIST	Ada Software Selection assISTant
BCD	Binary Code Decimal
BCNF/3NF	Boyce-Codd normal form/Third normal form
CASE	Computer Aided Software Engineering
DOD	Department of Defense
DSS	Decision Support System
GSC	General Software Characteristic
GUI	Graphical user interface
I-CASE	Ideal software development environment
IPSE's	Integrated Project Support Environments
KB	Knowledge base
MPW C	Macintosh Programers Workshop C
OOD	Object Oriented Design
SADT	Structured Analysis and Design
SAI	Software Area of Interest
SAME	Structured Query Language Ada Module Extensions
SD	Structured Design
SEA	Software Engineering Activity
SEE's	Software Engineering Environments
SQL	Structured Query Language
SQLDA	SQL Description Area
SSC's	Specific Software Characteristic's
STEM	Software Tool Evaluation Model
STEMdB	STEM Database
STSC	Software Technology Support Center
T&E	Test and Evaluation
TEG's	T&E Guidelines
TEP	T&E Procedure

AFIT/GCS/ENG/91D-04

Abstract

Information overload has long been a problem in the fast moving technical field of software development. Yet quality information is needed to make informed decisions about buying software tools that help in software development. Computer Aided Software Engineering (CASE) tools help to coordinate and control information in large software developments. Many CASE tool purchases, however, are being based on ad hoc tool evaluation and selection methods which depend on biased vendor information. To capture specific knowledge about how to pick a tool for a given software development effort, a historical database that identifies important tool characteristics needed to be maintained by an unbiased organization and a mechanism (in the form of a decision support system) for interpreting that database needed to be made available.

To address this deficiency, the Software Technology Support Center at Hill AFB in Utah was developing a CASE tool selection support tool, the STEMdB. This research accomplishes an analysis of this tool and suggests ways to make it more robust, portable and maintainable. It presents an object oriented approach to the design while addressing the issue of portability by accomplishing an Ada to Structured Query Language (SQL) abstract interface design.

ANALYSIS OF A DECISION SUPPORT SYSTEM FOR CASE TOOL SELECTION AND THE SPECIFICATION OF AN ADA TO SQL ABSTRACT INTERFACE

I. Introduction

Overview

This research was directed at comparing two known CASE tool evaluation and selection prototypes to be able to influence the ongoing prototype effort towards developing a more maintainable and upgradeable decision support tool. The first prototype tool was developed based on object oriented methodology, the second ongoing prototype is being developed using the functional decomposition method. The research also provides a suggested abstract interface design that would be used by the system to communicate with an Structured Query Language (**SQL**) database.

Background

When computers were first developed the engineering community had its hands full trying to optimize computer hardware to reduce high costs. Computers were so expensive and huge in the days of relay and vacuum tube technology, that engineers would never have been able to visualize the 1990 desk top personal computer. With this new, smaller more powerful technology, the costs of hardware became insignificant compared to its counterpart, software.

Engineers were forced to standardize the development approach to hardware as a result of those initial high costs. They neglected developing just such an approach for software development, since it was considered more an art than a science from their view-point. As a result, software was approached in an ad hoc way until the Structured Design (**SD**) Methodology surfaced. Once this methodology surfaced and was implemented more complicated, larger software efforts could be undertaken with more success. But as these efforts got larger, the SD methodology alone was not enough to ensure that software systems could be developed and implemented on time and within budget. This lack of control of software cost (which is the critical cost driver in today's systems) and software quality, resulted in what scholars of the 1980's referred to as the software crisis [21:389].

This is where Computer Aided Software Engineering (CASE) tools entered into the equation. Many in the software engineering community viewed these tools initially as a solution to the crisis. However, full understanding of the necessity of having a well understood software process model is a prerequisite to the purchasing of any CASE tool [20:41-44; 27:9]. It is also beneficial to have a general understanding of different software engineering environments and how CASE tools fit into these environments. Once all this has been accomplished, software professionals can select a CASE tool, a disjoint set of CASE tools, or an integrated development environment that supports project needs. In order to make the best choice of development tools, a survey of available tools and their capabilities must be requested and reviewed. Unfortunately, the volume of data available on the different capabilities of tools is both subjective and overwhelming. Efforts have been made in two complementary prototype tools, the Software Tool Evaluation Model (STEM) [27:6-8] and the Ada Software Selection assISTant (ASSIST) [24], to automate this selection process. It is at this point where this thesis comes in.

Problem

There is no way of selecting CASE tools or, in the bigger sense, software support environments, without extensive research of very subjective data. The human ability to process data effectively is taxed beyond its capabilities when juggling more than about seven plus or minus two pieces of information at one time. One CASE tool alone can have close to 100 different capabilities that must be evaluated [18:C.2-C.5]. Since humans are incapable of evaluating this much data, an automated evaluation system, that would scale the selection problem down, needs to be developed. At the time of this research, this researcher was not aware of any existing automation system that was able to implement evaluation criteria, take into account users' selection requirements, and propose solution tools.

The Software Technology Support Center (**STSC**) at Hill AFB, understood this problem and was developing the STEM prototype [27]. Although the developers of the STEM had not completely implemented the selection part of the prototype at the time of this research, what was there appeared to be tailored more to individual CASE tool selection than CASE environment selections. With the trend in CASE tools heading down the CASE environment path, the STEM may steer the CASE selection tool effort towards an obsolete end requirement.

The ASSIST prototype, however, is set up as a theoretical environment evaluator with the capability to look at sub-components of a total environment. Although targeted for Ada tools and environments, its principles are applicable to CASE tools in general.

Research Objectives

The objective of this research was to provide an analysis of an ongoing Decision Support System (**DSS**) design that would help guide the facilitators to develop a more substantial tool to meet future maintainability and upgradeability requirements. The potential for the STEM tool and its follow-on effort, the STEM Database (**STEMdB**) tool, to become highly sought after and used tools depends on their portability and flexibility. This research provides ways to achieve this portability and flexibility.

Assumptions

The initial research assumptions were:

- CASE tool decision support selection systems are badly needed in government and industry, and will continue to be needed for some time.

- The ASSIST theoretical architecture would provide a good basis for comparing the STEMdB tool's functionality.

- Providing both an object oriented approach and an abstract interface approach to the developing organization would help to convince them to design the STEMdB using these approaches versus using a functional design which was totally dependent on a single database.

- The underlying structures of an ASSIST and the STEM data representation could be made compatible.

- The SQL Ada Module Extensions (SAME) was designed using good software engineering principles so it was a good Ada to SQL binding method to choose.

- A STEMdB developed with the end goal of supporting remote users would force its design to be more robust, if it was designed using good software engineering principles.

- Providing a top level interface design would help convince the developing organization to approach the STEMdB design from a remote user requirements perspective.

- Providing a top level object oriented design approach to the STEMdB would help to convince the organization of it's merits and would help them to understand how to work with the STEMdB as a DSS that depends on a Knowledge Base.

Scope

The research analysis only concentrated on the portions of the software development life cycle domains that were targeted by the STSC. This researcher accomplished an analysis of the STEMdB design from the data object and behavioral perspectives. This researcher also accomplished a top level object oriented design of the STEMdB while providing information on how to work with a knowledge base. Finally, this researcher accomplished a top level abstract interface design using the SAME method. The applications that call the interface resources were not provided.

Approach

First an extensive literature search was accomplished to get a background on decision support systems, CASE tool categorization, CASE evaluation and selection criteria, database modeling, Ada to SQL bindings, and the Macintosh development environment. In Chapter III all this knowledge is used to evaluate both a working prototype and a proposed follow-on prototype effort. Chapter IV addresses solutions to two related problems with the STEMdB design that were identified in the analysis. In particular, it presents an object oriented design and an abstract interface design. Finally Chapter V concludes with lessons learned.

II. Literature Review

Introduction

The following literature review was accomplished to learn about CASE tools, Software engineering environments, CASE evaluation and selection processes, CASE selection system design, Entity Relationship modeling and a Macintosh development environment.

Specifically, the literature gave insight into reasons an organization would purchase CASE tools, and identified different ways to categorize CASE tools. Some articles discussed the pitfalls of CASE while others addressed the different levels of tool integration. The Entity Relation diagramming, and the Decision Support tool studies helped with analyzing the STEMdB design and suggesting improvements. The Macintosh HyperCard development environment literature also provided insights into how the ASSIST prototype accomplished its job and into how a commercial database, Oracle, provided a programming interface to its database.

CASE Tools, Why Bother?

Brook's book, [12], as well as Batt and Sims in [10:1; 28:1] discussed different aspects of the software crisis. Both Batt and Sims theses [10; 28] used the perceived crisis as justification for studying CASE technology as a possible solution to the crisis. Other literature also eluded to aspects of the software crisis as reason to study the possible benefits of CASE in software development.

Software tool automation could facilitate consistency checking, automatic documentation of design, configuration management of code, overall project tracking, design generation, the enforcement of formalisms and structured methodologies [26:75], and design accessibility (via tools that use relational database queries or Hypertext

[11:23]) according to general consensus. Hypertext was defined by Bigelow in [11:23] as :

> **Hypertext is a medium-grained, entity-relationship model that lets information be structured arbitrarily and keeps a complete version history of both information and structure.** [11:23]

CASE Categories

CASE tools were categorized by different authors in distinct ways. Tamanaha viewed CASE tool categories as falling within some phase of the Waterfall software life cycle model, as falling within a narrow discipline like documentation, or as falling into a specific application domain like real-time systems [29:6]. She identifies five phases associated with the Waterfall model: requirements/analysis, design, code, test and maintenance.

Sim's took a much higher level approach than Tamanaha. He viewed CASE tools as falling in four categories: Upper-CASE, Lower-CASE, Reverse Engineering/maintenance-CASE and Project Management-CASE. The Upper-CASE tools applied to the life cycle phases requirements through design; the Lower-CASE applied to code through test; the Reverse Engineering/maintenance-CASE applied to the end of the life cycle after test; the Project Management-CASE applied to all phases of the life cycle [28:33-34]. Sim's also reported survey results which highlighted Upper-CASE tools as being the most used, 75% usage in industry during 1988 versus less than 20% usage in any other of his categories [28:72].

CASE Pitfalls

In both [22:25-29; 20:41-44] the reader was warned about potential pitfalls that could occur while attempting to begin and to continue using CASE technology. Not understanding an organization's software process model was one pitfall that was repeatedly acknowledged in different literature. Keuffel reinforces this by ending his article with this quote about the software methods:

> **"CASE systems," DeMarco concluded, "have tended to be most helpful to people who had the least trouble applying the method, even without automated support."** [22:29]

Another identified pitfall was the fact that vendors were producing volumes of subjective material about the sanctity of their products [22:25]. Keuffel in [22:29] stated that "current CASE tools either ignore many rules or enforce them too rigidly". He advocated making tools "transparent" to the users [22:28] in order to support user friendliness.

CASE Tool Integration

Different levels of CASE tool integration were also a common topic in the literature. According to Keuffel, the highest achievable level of integration was achievable through "Groupware" [22:29]. Groupware is a real-time response CASE system that could become a design group's intelligent whiteboard.

Batt, in [10:13], introduced the term **I-CASE**. He defined this term as, "I-CASE toolkits incorporate all the best features of many CASE tools into a single package that is intended to cover the entire SDLC" [10:13] (where **SDLC** stands for software development life cycle). Batt stated that a true I-CASE tool would incorporate over 100 sub-tools and that no true I-CASE tool existed at that time. The term I-CASE can be thought of conceptually as the ideal software development environment where all tools inter-operate and share common data.

The term, Data Encyclopedia, was introduced and defined by Batt as "The heart of an I-CASE toolkit." [10:13]. In other environment contexts it was called the object management system. The Data Encyclopedia's counterpart in the non-integrated CASE world was the equivalent of the Data Dictionary. Batt indirectly addressed at least four levels of CASE integration in his thesis. Those levels were: no integration, limited integration in one CASE tool line, some integration outside of one product line and total integıation. The integration literature, in general, agreed that higher levels of CASE integration could be achieved from industry support of open CASE architectures and CASE tool interface standards.

Software Engineering Environments

Wybolt in [30:57] provides a good overview of the requirements/attributes of Integrated Project Support Environments (**IPSE's**). He lists the following as desirable characteristics of an IPSE [30:57]:

- cover all phases of development and support
- support or connect with multiple disciplines like CAD or CAE
- cover many project activities such as planning, controlling and documentation
- deal with multiple vendors and platforms
- adapt to existing organizational culture and work flows
- can be introduced incrementally
- can be serviced and supported
- are fast and inexpensive

Wybolt also clarifyed how a framework was the building block of IPSE's. He defined the structure of a framework as having a common user interface, a set of tools, an integrating agent, and an object management system. He defined the four styles of integration that could be implemented within a framework as: presentation, control, data and semantics. Presentation integration was provided through a common user interface; data integration among tools was provided through a common repository; control integration was provided through repositories with links between them; and semantic integration was provided through modeling of the semantics of tools and their data along with conflict reconciliation. [30:56-59]

Finally, Wybolt clarified that "integration is a property of a relationship between two or more tools" [30:59] not a property of an environment or tool.

CASE Evaluation and Selection Guidelines

The draft document in [1] provided some guidance on the process of evaluation and selection of CASE tools that support the software project management, configuration management, and software engineering domains. To clarify the difference between the two processes, evaluation is "the process of measurement" and recording while selection is "the process of applying thresholds and weights to evaluation results and arriving at decisions" [1]. The IEEE document in [1] also emphasized the iterative nature of these two processes and the necessity for feedback from the selection process. Feedback was to take the form of user critiques of any selection and evaluation system that was being used. Without this feedback neither process could become a mature process. Criteria that are measured both quantitatively and qualitatively are the common link between the two processes. [1] defined users requirements as the only driving force towards deciding on these criteria and suggested several possible classification schemes, including the E&V Reference manual [2].

An evaluation process should address tool issues of multi-project support and multi-function support. In other words it should provide for an evaluation which measures tools that are abstract enough to address both the immediate project functional needs of the user as well as any future new project requirements. It should be repeatable for the objective criteria and it should accommodate documentation of multiple evaluator inputs for the subjective criteria. [1]

A selection process should allow for a narrowing down of information. The reasons for steps taken during the selection process should be recorded. Since selection criteria will be either numeric or binary, a way of weighing and combining both types must be addressed. Traceability of the feedback loop to the evaluator process must be addressed. If the evaluator process never receives feedback from the users or tool selectors, then the evaluator works in a vacuum and the whole evaluation and selection process runs the risk of being useless. Finally, a sensitivity analysis should be supported. Sensitivity must be addressed so that selector or user is able to judge how valid his /her results were based on the limitations of the system. [1]

The workshop notes in [3] discussed pre-selection process strategy considerations that need to be considered prior to an organization entering into the CASE tool selection process.

CASE Selection System Design

The Lawlis dissertation provides guidelines on how to use the above evaluation and selection techniques, as well as other design criteria to create an automated decision support system for CASE tool selection.

The basic processing in her design began with the narrowing down of candidate tools. She assessed three levels with respect to the user's needs: the scope/context of the solution space, the tool category, and the application area. Once these were input her tool requested the specific weights of applicable characteristics. Finally the ASSIST would process the inputs and provide some suggested solutions.

The basic structure of systems that support this design would be composed of four top level objects as is shown in Figure 1.

The Knowledge Acquisition object allows users to load and update data stored in both the Knowledge Base and the Database. The user could access the Knowledge Base and the Database indirectly through both this object and the Decision Logic object.

The Knowledge Base object provides the operations necessary to use the data. It could accomplish these tasks because it maintained the design knowledge on how the data was structured. In addition, it could control how a selector used that data.

According to [24:70] a critical part of the design of a DSS is the isolation of its dynamic elements and operations to the Knowledge Base Subsystem. Specifically, the

Knowledge Base Subsystem should define which software characteristics are made visible to the rest of the DSS. It should know the types and the allowable operations on all characteristics and it should provide this knowledge to the rest of the system. Since leveling of information is necessary to make a selection tool effective, the Knowledge Base Subsystem should contain knowledge on levels of views and it should provide the information on how to process these levels to the rest of the system. Finally a Knowledge Base Subsystem should contain the knowledge of how to transform evaluation data into ratings. By concentrating all of these volatile design areas, a DSS design maintains robustness and ensures that most future revisions affect only the implementation part of the DSS Knowledge Base object.

One concept of this design that was hard to comprehend was understanding the difference between a Knowledge Base and a Database. It was discovered that the differences between a database system and a knowledge base system could be distinguished by the kind of data stored in each and the operations provided by each. For example, a database system would provide the services of storage and retrieval of all evaluation/selection data. A knowledge base system would provide for the interpretation of that data.

The Decision Logic Object is the controller of the Decision Support System processes. Essentially, it addressed requirements that were based on a user that was in the process of selection.

Entity Relationship Diagrams

A quick explanation of an Entity Relationship diagram and how it relates to the table relational design is provided next, so that readers will have the proper background to understand the work accomplished in Chapter III. Entities are objects that exist and which have specific descriptive attributes that distinguish them from one another. For instance a mother and a child are two entities which both have an attribute of social security number, but the value of those attributes differs for each of them. A relationship is a connection between two or more entities. For instance two relationships, has_child and has_genetic_child, between a mother and her five children would contain the names of all five children in the first relationship and would contain the name of only one child in the second relationship if the mother had adopted four of her five children.

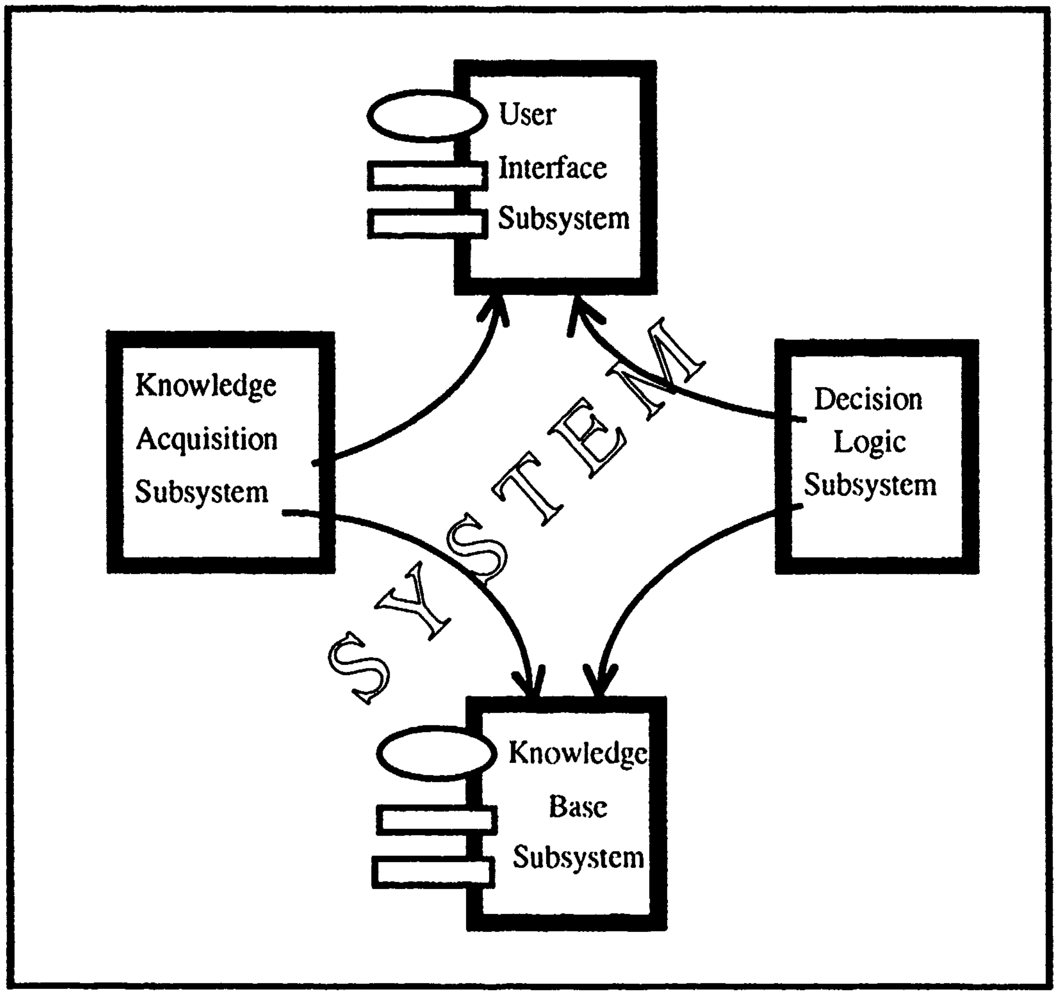

Figure 1. Decision Support System Top Level Design[24:291]

Entity Relationship models provide several different pieces of information. They provide entity and relationship definitions in the form of named rectangles and named diamonds, respectively. They display all important attributes (distinguishing key attributes with bold capital letters), and they display the cardinality and optionality (these are explained in the next paragraph) between entities and relationships. They also support the notion of abstraction by the concept of an aggregate. In simple terms an aggregate is a means of showing a n-way relationship (where "n" is the number of related entities). In structural terms an aggregate is a super entity that encloses two or more related entities and acquires the key attributes of those related entities.

Cardinality and optionality are constraints that the design of an actual database implementation would maintain [23:28]. Cardinality is just a mapping function between two related entities. For instance, the cardinality could define that exactly one of each entity relate to one of another (a one to one function) or it could define a one to many, a many to many, or a many to one relationship. Optionality shows an existence dependency between two entities over a relationship. If an entity has a mandatory relationship with another entity, then the latter entity will need to be altered as a result of a deletion in the former entity (in one to many/many to one, but not necessarily in many to many). Conceptually, optionality can be understood by asking the question, what is the minimum number of associations that must hold in a relationship between entities. An existential dependency of one entity on another means that the dependent entity does not make sense in the model if its required entity relationship no longer exists. Finally, the graphical representation of optionality and cardinality takes the following form: "α:β". Where α represents the optionality and it takes on values of either "0" or "1" and β represents the cardinality where it becomes a letter in the alphabet to represent "many" or it becomes the number "1". For instance, "1:N" means an entity is mandatory and that it has the cardinality of "many" in a relationship with another entity.

HyperCard

In [2:11] it was stated that HyperCard could be used to "gather, organize, present, search, and customize information". Design ideas based on HyperCard's data representation were presented in [1]. HyperCard supports external function calls to pre-compiled C and Pascal code. This tool was used to create a user's tutorial for each of the three prototype tools in [7; 8; 9]. It was also used as a development environment for [24] and as a database interface called Hyper*SQL within Oracle [25]. HyperCard provides the user interface and test program capability which reduce the level of work required for a prototype effort in the Macintosh environment.

Conclusions

CASE tools are needed to help in both large and small software projects. The larger the communication gap in a project the more severe the problems become. CASE tools as they become more mature, and as they integrate more, should achieve some positive impact on these software problems by increasing project data accessibility.

There are multiple ways of viewing CASE tool membership categories. There are also several ways of getting into trouble with CASE tools. Integration appears to be where the future of CASE development is headed.

III. *Description and Analysis of The STEMdB Prototype Design*

Overview

This chapter provides the reader with an understanding of Software Tool Evaluation Model Database (**STEMdB**) design efforts that were going on concurrently with this research. It begins by providing a description of the the Software Technology Support Center's (**STSC's**) development efforts and mission objectives. It discusses the background work that the STSC had accomplished which lay the foundation for development efforts towards the STEMdB. The chapter then provides the reader with the objectives of the STEMdB within a defined scope. It then accomplishes a multi-level description of the design followed by an analysis. The analysis results were based on the knowledge gained from Chapter II. In particular, the analysis compares the STEMdB design to the knowledge gleaned from [1] and the ASSIST prototype.

Description of STEMdB

Scope and Purpose of STEMdB. Scope: To define an evaluation framework that could be populated with evaluation information that would support the selection of CASE tools. Specifically, to support selection of CASE tools that would meet support criteria over multiple phases of the software development life cycle, different application domains and different subsets of software engineering activities.

Purpose: According to [19:1], "The purpose of *stemDB* is to allow the systematic, reliable, repeatable, and helpful selection of CASE tools for those in need of such technology".

Background of STEMdB. In order to understand the background of the STEMdB design at the time of this research, the historical work of the organization that was developing the STEMdB had to be presented. The STSC located at Hill AFB in Utah was the developing organization. Part of the mission of the STSC was to provide the Air Force with "centralized support for the evaluation and selection of software tools, methods and environments..." [18:1]. To accomplish this mission, the STSC defined an iterating process which identified both software problems and requirements of the Air Force, analyzed current software tools and technologies, and recommended possible solution tools, methods, and environments [18:1].

According to [21:5] "the primary objective" for any process should be "to achieve a controlled and measured process as the foundation for continued improvements". Humphrey showed in [21:5] that the first step towards achieving a mature process at this level would be to achieve repeatability. To achieve repeatability, the STSC implemented their newly defined iterating process within the framework of a Test and Evaluation (T&E) Process. This process was composed of area-specific T&E Guidelines (**TEG's**) that were used to evaluate software characteristics. Given a tool, a software characteristic and an area of interest or domain as inputs, the T&E Process would produce two outputs: an evaluation result and a T&E Procedure (**TEP**). A TEP was the documentation of an evaluator's actions taken while implementing a TEG. Figure 2 illustrates this process.

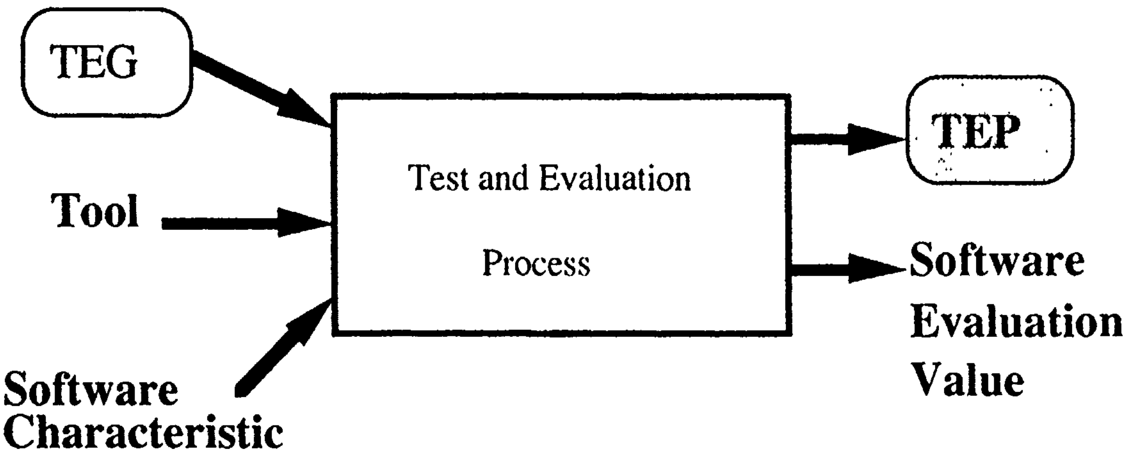

Figure 2. STSC Test and Evaluation Process

The culmination of these efforts, at the time of this research, was a series of databases that categorized specific software characteristics within high-level functional domains of software usage (i.e., Test, Documentation, Upper Case, Software Engineering Environments (**SEE's**)). The STSC defined these high-level domains as tool domains where "Tool domains categorize software tools by their major functional capabilities to compare similar tools" [18: 7]. Figure 3 gives a clearer picture of how to visualize these domains. The high-level domain equates to the Evaluation Framework node in the tree of Figure 3. The rest of the nodes in the figure represent the software characteristics that would be analyzed for a specific tool. Appendix A provides a representative STSC listing of specific software characteristics and qualities for the Requirements Analysis and Design (or Upper Case) Domain [18:66-75].

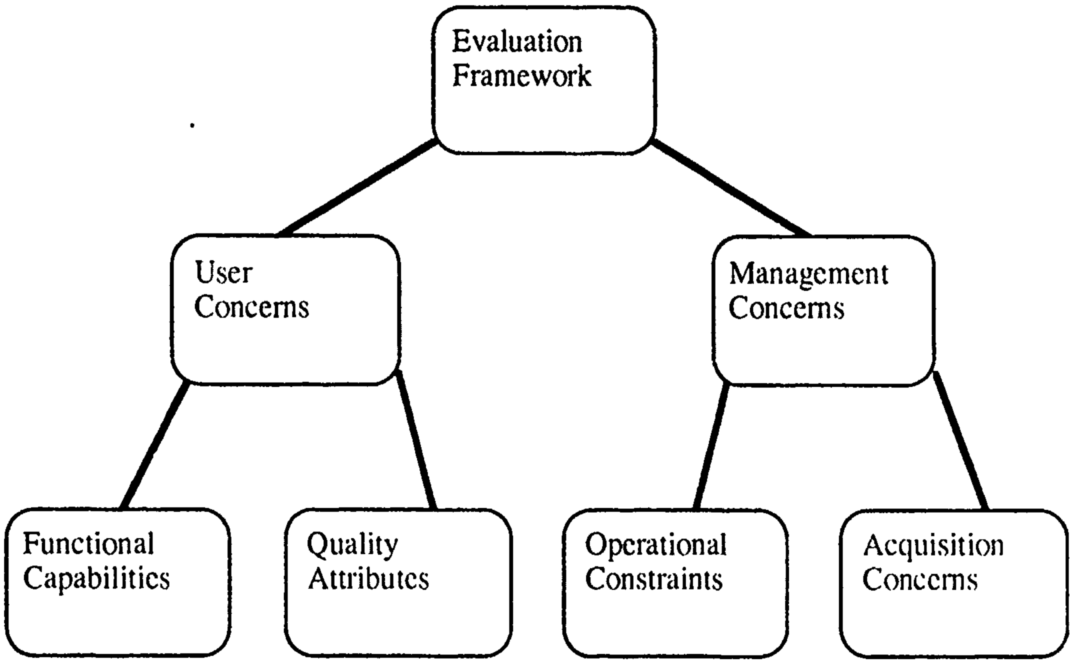

Figure 3. Example Tool Domain Evaluation Framework [18: 12]

As the reader will soon realize, all of this work helped the STSC to define the specific requirements that an automated system, the STEMdB, should be built to meet. With this background in mind, the details of the STEMdB design can now be provided.

Top Level Description of STEMdB. In order to describe the functional operations of the STEMdB, the structural and functional requirements will be presented next, followed by an implementation design description.

Representative Requirements Listing. The list is a representative rather than exhaustive list of key STEMdB requirements as recognized by this researcher. For more details the reader is referred to [19].

1. Build STEMdB tool to work around a hierarchical organization of CASE tool software characteristics.

2. Use linearly weighted combination of a tool's characteristics to arrive at a scalar measure for tool scores [19:8].

3. Use the identifying concept of a Software Area of Interest (**SAI**) to categorize tools. The SAI is defined by a high level domain, a target application or Software Engineering Activity (**SEA**) within that domain and a life cycle phase. The STSC defines a domain as being either "Management, Development, Test, Review, or Product Support". The STSC provides an example of an SEA in Management as being either "Project Management" or "Configuration Management". Finally the STSC defines a life cycle phase as consisting of one of the following: "Concept, Requirements, Preliminary Design, Detailed Design, Implementation and Unit Test, Integration and Test, Acceptance and Delivery, or Maintenance". [19.5]

How this all relates to the "high-level functional domains of software usage" is defined by the part of the STEMdB called the Formulator (a description of the composition of the STEMdB is presented later in this chapter).

4. Compute a function related score for each characteristic in the data model using the linearly weighted method. Perform a similar analysis for each of twelve quality attributes associated with the characteristic. Allow the user to enter the weights used. [19:9]

5. Use the framework of a General Software Characteristic (GSC) with instantiations of this framework, Specific Software Characteristic's (SSC's) containing a functional value, a TEP, quality values and evaluation information for specific tools. All GSC's will be defined by the functional/non-functional tooi characteristic identification work being accomplished at the STSC during the time of this research. A more detailed definition of a GSC will follow in the implementation and detail level design descriptions.

6. Allow five types of evaluation answers (Evaluate Children, Yes or No, Multiple Choice, Single Item Checklist, and Text) within the characteristic framework [19:12].

7. Support three separate functional operations, Formulation, Evaluation and Selection. Allow concurrent evaluation and selection operations, but force the formulation stage to a stable state before allowing evaluations and selections to occur. [19:10]

8. Use a commercial database that supports Structured Query Language (SQL) to manipulate and store the characteristic data. Provide for a tool interface that will issue SQL commands to the database and receive data from the database. The commercial database must be able to support up to 2000 tool evaluations with as many as 1000 software characteristics. [19:10, 20-23]

9. Front end programming language chosen for implementation will be "limited to those that support SQL commands with a particular database" [19:75].

10. "Platform upon which the system will run is limited to those that can host both the front end implementation language and the database" [19:75].

Implementation Design Description. Essentially the STEMdB is composed of five systems: The commercial database, the Formulator, the Evaluator, the Selector and a user interface front end. Figure 4 shows how such a system could be visualized. The arrows indicate communication between components in the form of commands.

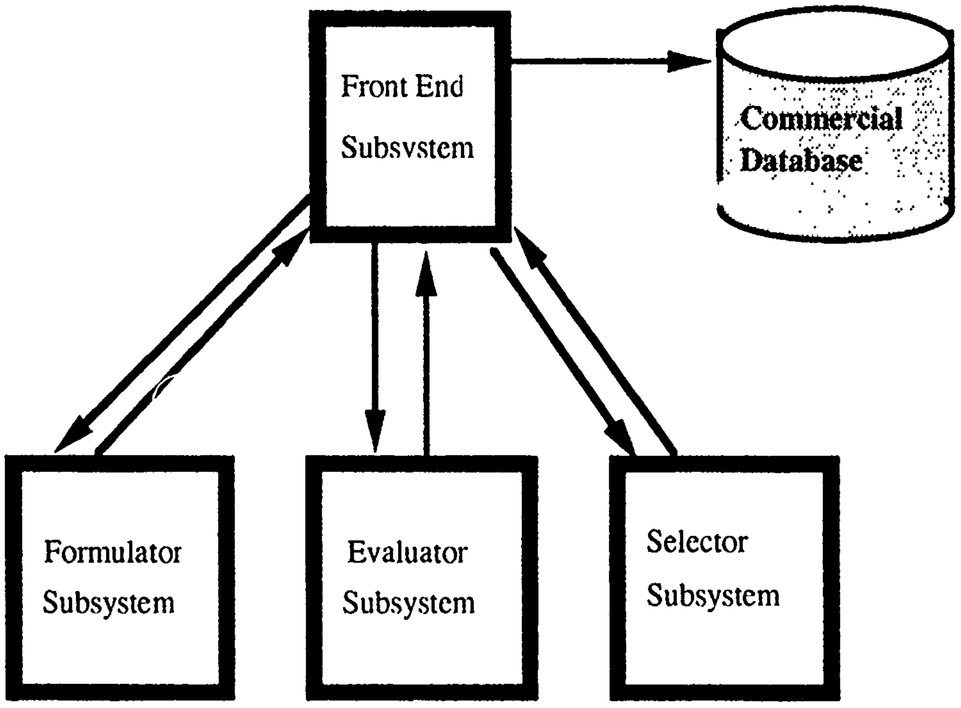

Figure 4. Original Design Components

Conceptually, the data that describes a CASE tool's functionality within a domain can be viewed as a description tree, where the functionality of the top node of that tree describes the functionality of the tool. This concept is based on requirement 4, where the functionality of a node is equal to the weighted sum of the functionality of its children. Each node in the description tree will be a SSC which contains the evaluation data associated with the functionality of that node.

The database subsystem would be chosen to provide both an SQL interface and the full functionality of a standard database. A front end interface would be used to communicate between the database and the other three subsystems.

The Formulator's principle function would be "to specify the various functional areas that describe the performance of CASE tools and then to build a description tree for each area" [19]. The Formulator subsystem must be able to build description trees out of a framework of GSC's. The GSC's chosen for the framework will be identified as a result of the work that is still ongoing at the STSC under the T&E Process. The method that describes how to evaluate a description tree will be implicitly defined by the "type" of evaluation answer that the Formulator encodes at each node in the tree. The Formulator needs to be able to access the database to store SAI design's.

The Evaluator must be able to access a specific SAI design in the database, to understand how to evaluate its description tree based on the types associated at each GSC node, and it must be able to store the evaluations as an instantiation of the SAI design for a specific tool. Once the Evaluator subsystem evaluates a GSC, it should be able to store the results in the SSC that maps to that GSC.

The Selector must be able to access tool specific instantiations of the SAI design. It must be able to define weights for every SSC node in the design. It must be able to save these weights in a weight set area of the database for the tool specific instantiation. It must be able to understand how to score a tool and its individual characteristics given both an SAI description tree and a weight set. It must be able to save sets of user defined tool names.

Detail Level Description of STEMdB. The next three sections present a description of the STEMdB design data semantics, subsystems communications, and subsystem processing.

STEMdB Data Model Description. One of the best ways to visualize a data model is through a graphical illustration. Data models provide both "data and structural information" [19:3]. The requirements document in [19:25] provided the table relational diagram shown in Figure 5, but this illustration did not provide an immediate picture of how the data was structured without extensive study.

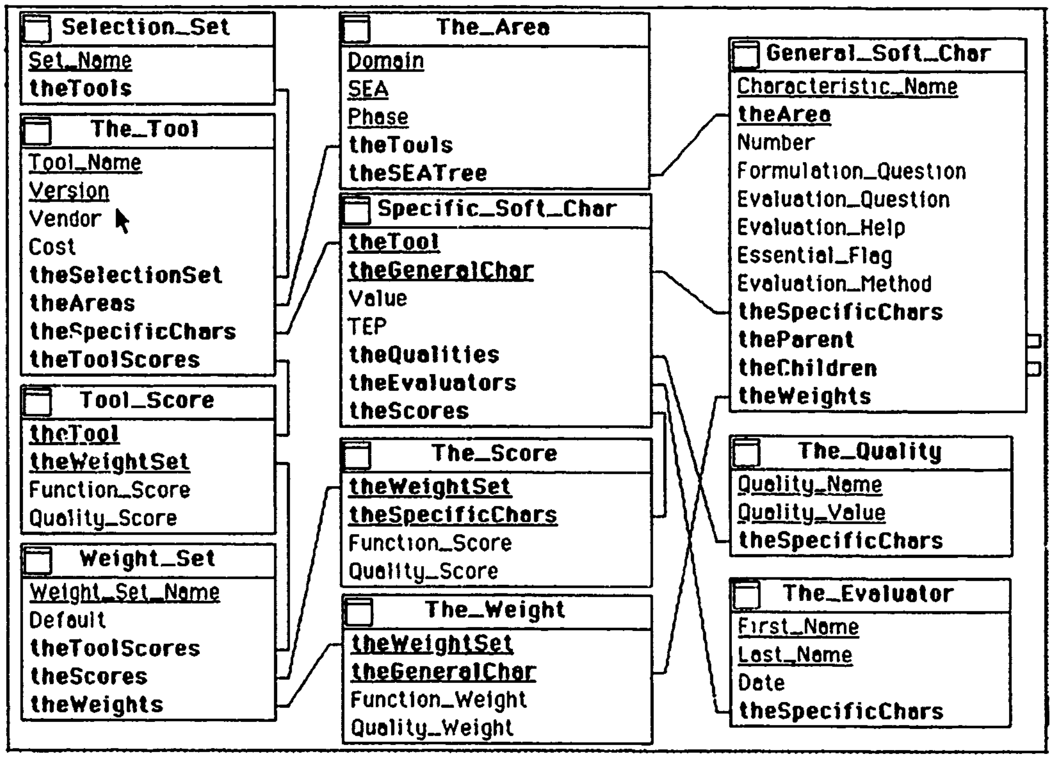

Figure 5. STEMdB Requirements Table Relational Diagram

To gain more insight into how the data was structured, the entity relationship (ER) diagram in Figure 6 was developed from both general tool usage knowledge and from the table relational diagram of Figure 5. An attempt was made to maintain the same naming conventions used in [19:25-32] in order not to confuse those referring back to this STSC requirements document. To achieve better readability and understandability of the design, however, the following changes were made :

1. Four relational table names were changed
   - Specific_Soft_Char and General_Soft_Char tables were expanded to their full names.
   - The_Score table was renamed software_char_score
   - The_Weight table was renamed software_char_weight

2. The linking relationships were provided shorter names that all began with "link" and ended in two capital letters that matched the first distinguishing letter of each linked table. For instance linkAT connected the The_Tool table to the The_Area table. There were two cases where this rule was broken. The first case involves all non-trivial relationships (relationships that contained attributes in excess of the mandatory keys) of linked entities. All non-trivial relationships were given a descriptive relationship name. The other case involves the "root_node" relationship. This relationship needed a more descriptive name to identify it as linking an area to the top node of a description tree.

3. Finally, one linking relation was added between The_Area and The General_Software_Characteristic tables. The root_node link was added to clarify the relationship between the Area and a general software characteristic.

The way that an ER model is converted to the table relational form is straightforward. In general all entities and relations become tables with their attributes becoming table field names (or column names in other words). Distinguishing attributes become key attributes in the tables. Each general characteristic, for example, would be designated a row (or a record) in the General_Software_Characteristic table. To access the attributes of a specific general characteristic, one would search for the unique key, GCS_ID, that equals the desired GCS_ID key. Relation tables also inherit as key attributes the key attributes of the entities that they connect. If aggregation exists then the aggregate inherits attributes of its internal relationship and it becomes a table. For instance, the Specific Software Characteristic aggregate table would have field names of:

SSC_ID, GCS_ID. TOOL_ID, value, and tep. The SSC_ID, value and tep were already attributes of the Specific Software Characteristic relationship , it inherited the TOOL_ID key from the The_Tool entity and it inherited the GCS_ID from the General Software Characteristic entity.

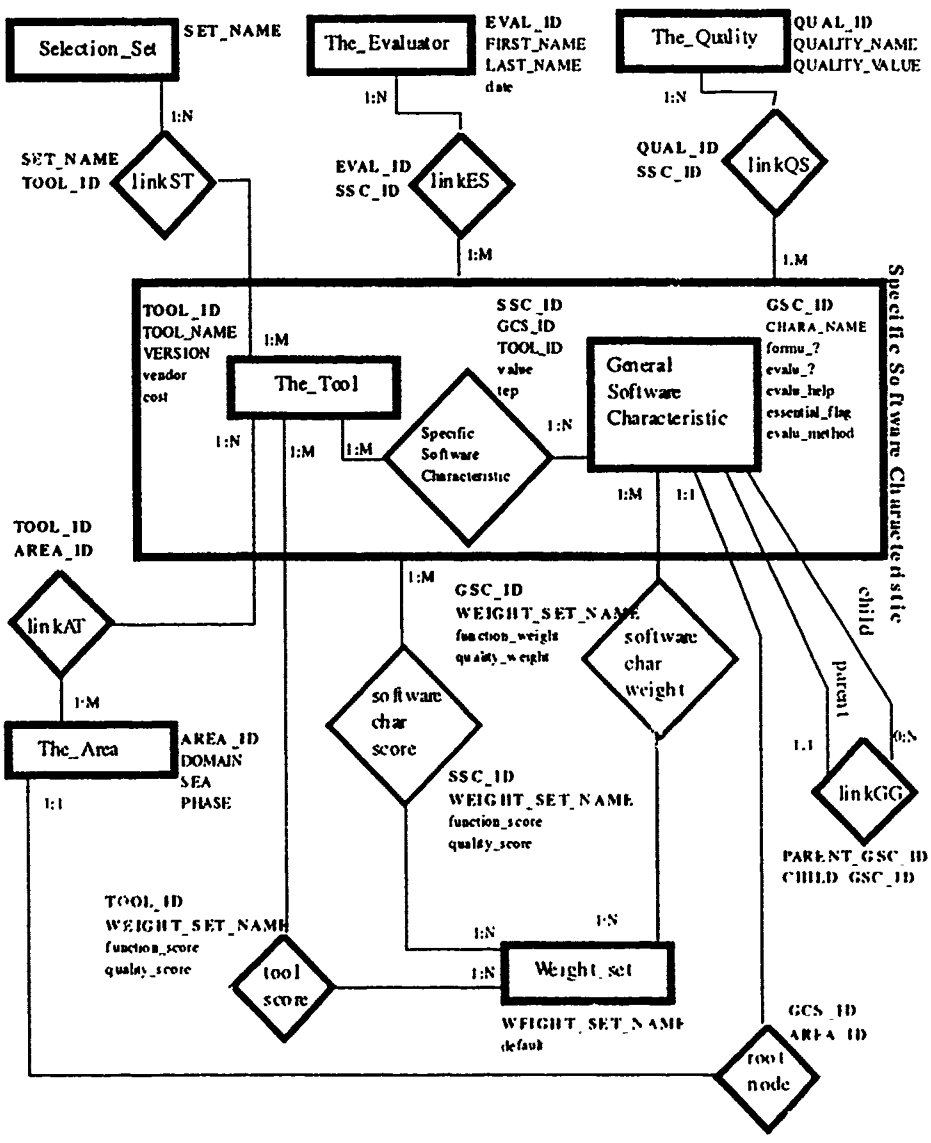

Figure 6. Entity Relationship Diagram of STEMdB

Description of Key Entities and Relationships of Figure 6. The entities in the STEMdB data model are as follows:

1. The_Tool - Contains basic tool information as defined in the attributes of Figure 6.

2. The_Area - Functional areas to which CASE tools will map. These areas map to the high level tool domains discussed earlier.

3. General_Software_Characteristic - Framework which the STEMdB tool is built on. The following is an explanation of all non-self explanatory attributes:

formu_? - Is an explanation comment stored in the GSC by the designer of the domain description tree. The comment explains why the designer chose to insert a particular characteristic at a particular location in the tree.

evalu_? - Is the evaluation question that an evaluator must answer to evaluate the characteristic.

evalu_help - Explains the justification for addressing the evaluation question of a characteristic.

essential_flag - Maps back to the STSC "short tool list" concept. In [27] a short tool list was the list of tools that met the minimum Air Force Requirements as determined by the STSC. In the context of the STEMdB, the Formulator tool flags a characteristic as essential when it must be evaluated favorably for the tool to be considered in a selection.

evalu_method - This is a Boolean variable that lets the evaluation tool know whether or not this characteristic must be evaluated. When a characteristic contains the type of "Text", for example, this attribute will have truth value of "false". All other answer types will result in a truth value of "true".

4. Specific_Software_Characteristic (SSC) - This is the instantiation of the GSC after it has been evaluated. The "value" attribute contains the functional results in a form that is defined by the answer type in its corresponding GSC. The "tep" maps to the TEP of Figure 2 and represents the same information.

5. Weight_set - Contains the names of stored weight sets for specific high level tool domains. The default attribute is a Boolean that determines if this weight set is a default weight set.

6. Selection_Set - Contains the names of sets of tools that were considered and stored by a selector.

7. The_Quality - This entity has a quality_name attribute that can be assigned one of twelve quality names (see Appendix A). The QUALITY_VALUE associated with a specific QUALITY_NAME is an integer score (between 0 and 10) arrived at by an evaluator based on a TEP.

8. The_Evaluator - Represents specific evaluator information about the person who evaluated a SSC for a tool in a given functional domain.

The non-trivial relationships (non-trivial meaning that the relation tables contain attributes in excess of the mandatory inherited key attributes) are as follows.

1. tool_score - Given a weight set and a tool this relation provides the overall function and quality score of a tool within the context of a domain.

2. software_char_score - Given a weight set and a SSC this relation provides a function and quality score for that particular SSC within the context of a domain.

3. software_char_weight - Given a weight set and a GSC this relation provides the specific function and quality weight for the GSC.

STEMdB Interface Design Description. There were two types of interfacing that [19] described, the user interface and the functional tool subsystem interface to the database subsystem. Although the user interface was not required to adhere to any particular graphical user interface (GUI) standards, the design did specify that a GUI would be used. The tool interface between database and STEMdB application was defined in [19:21] as in Figure 7:

STEMdB Functional Design Description. A top level description of the functional requirements was already provided earlier in the Implementation Design

Description section. The reader is referred to [19] for a detailed description of the functional components of the design. A flavor for these details will be provided in the analysis section of this chapter which follows.

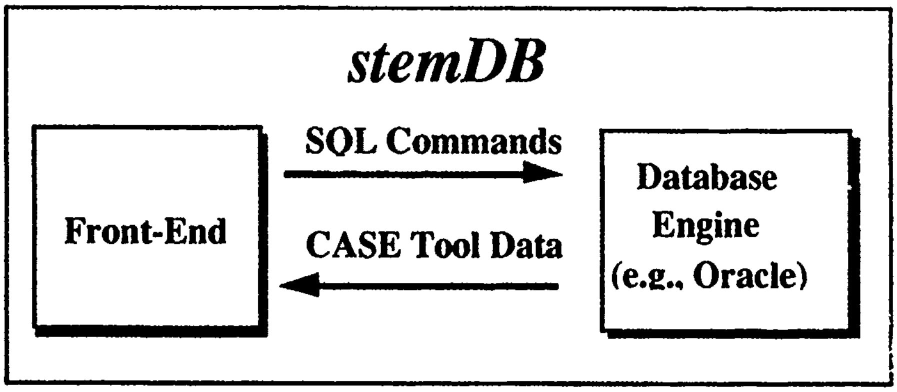

Figure 7. STEMdB Basic Components

Analysis

Areas Well Designed. In general the STEMdB effort had accomplished a lot of good work towards creating the end product of the STEMdB tool. The developing organization understood the need to learn more about the problem space and used prototyping as a means to help firm up requirements. To accomplish this prototyping, two prototyping efforts were launched. The first effort resulted in a Think Pascal implementation that was developed in the MacApp development environment. This effort produced the three working prototypes referenced in [8; 7; 9]. These prototypes concentrated on modeling Formulator and Evaluator tool functionality more than the Selector tool functionality. Another product of this prototype stage was the creation of three Hypercard Tutorial stacks that were targeted towards training users of the prototypes. The next stage prototype[19:3] was an ongoing effort that was described in the requirements document. Its purpose was to show how the STEMdB would work with a database storage facility, and to show how the user would interface with a STEMdB tool.

Much of the functionality described in the requirements document and the prototyping efforts showed the developing organization's dual concerns for providing both a useful tool and a user friendly tool. The following section identifies some of these requirements.

The design addresses how to process data from multiple evaluators by merging evaluation trees. The design also specifies that a Graphical User Interface will be used. The design partially addresses the issue of helping the user by providing textual information on the Formulator and Evaluator processing. The design identifies that a database could contain incomplete information from both Formulator and Evaluator processing. Incomplete information in this sense, however, is an added functionality of these two tools, since a user is released from the requirement of having to complete a session in one sitting. The design also addresses one aspect of maintainability from the viewpoint of supporting a rapidly changing data model. Since identifying characteristics of tools could change frequently (due to the ever-increasing improvements to software design), the design will address how an old data model can be mapped onto a new design [19:12]. Although this mapping concept was identified in the requirements document as necessary, no design was presented at the time of this research.

Areas That Require More Work. Even though the prototyping efforts produced many valid design requirements, questions still remained about the design's maintainability and upgradeability. This researcher identifies two top level areas that require more work due to a lack of maintainability and upgradeability in design considerations and unnecessary data and functional limitations. The first area identified discusses new approaches to designing the STEMdB tool that support upgradeability and maintainability requirements. The second area identified addresses improvements that can be made within the context of the present tools data a unctional designs.

New Approaches to Designing the STEMdB Tool. This section is divided into two new approaches for the present design. The first approach presents an argument for a change in design methodology, the second approach presents an argument for setting a higher goal for system maintainability and upgradeability. Both arguments complement one another.

A Change in Design Methodology. The STEMdB was designed from a functionally oriented approach. In the STEMdB requirements document the Front End module was identified as providing all the functionality of the STEMdB tool. It controlled the Formulator, Selector and Evaluator, while providing scoring, reporting, interfacing, database initialization, and a user interface [19:22]. Since the STEMdB was designed with behavioral goals that map closely to the behavioral goals of the Lawlis dissertation tool ASSIST, the STEMdB tool is a decision support tool. Lawlis states that there is "a strong relationship among the object oriented concepts and knowledge representation concepts of frames and semantic nets" [24:55]. She further states that if a decision support tool "uses these knowledge representation concepts", it should naturally follow that the tool be developed using object oriented concepts [24:55]. The STEMdB is developed based on the idea of frame based knowledge and semantic nets. The description tree is a semantic net and each node contains a frame of knowledge. Therefore, it should naturally follow that the STEMdB be developed using an object oriented design approach instead of the present functional decomposition approach. In addition, if the STEMdB is a tool that will be utilized over many years then upgradeability and maintainability become an issue. In the functional decomposition designs of past and present, maintainability and upgradeability are hampered by designs whose state information is dispersed throughout the functional modules. Borland is one example of a commercial company that has switched to object oriented programming in all of their products and is reaping the rewards of this new methodology. In an October 1991 US News and World Report news release, Borland claimed that the Object Oriented Design (OOD) methodology cut new upgrade release development time one third to a half while also reducing lines of source code. In the article, Borland used their old way of doing business, Structured Analysis and Design (SADT) and Functional Decomposition, as a benchmark for these assertions. Changing the design approach could reap similar rewards for the STEMdB when it comes time to maintain and upgrade it. Besides, much of the work that has already gone into developing the STEMdB prototype efforts can also be used in an object oriented approach.

The Lawlis dissertation lays the framework for an object oriented approach. Chapter IV shows how the STSC design can be re-accomplished using this framework.

Improving System Maintainability and Upgradeability. According to the design, the STEMdB was targeted for one platform that supported one type of SQL database. Furthermore, the STEMdB implementation language could not be Ada since requirements nine and ten in the Design Description Section of this chapter explicitly stated that the implementation language must be supported by a database provided interface. At the time of this research there were no known SQL commercial databases that provided interfaces in the Ada language. By eliminating Ada as an implementation language and by restricting the tool to operation in one specific environment, the developing organization was not providing for long term maintainability or upgradeability.

Ada provides many of the capabilities that support good software engineering practices which, when properly implemented, can create a more maintainable application. The language provides strong typing, the mechanisms for information hiding and the ability to communicate with other implementation languages. It also provides the capability to create abstract interfaces in software applications which can be used to establish hardware independence. For instance, when a software application using an abstract interface must be re-hosted on a different machine, the implementation part in the body of the interface will need to be revised but the rest of the application logic and structure will remain unaffected. Since Ada can communicate indirectly with SQL databases through pragmas, it becomes a candidate implementation language for the STEMdB. Further, since the Department of Defense (DOD) edict states that all DOD software developments will be accomplished in Ada unless it is not cost effective [4], the STEMdB must be developed in Ada . Using Ada to communicate with a database opens up a further design decision as to how to approach this interface design. At the time of this research much of the procedural program interfacing with databases used a de facto standard of pre-compiler technology [14:3]. Pre-compiler technology was a method used for embedding SQL statements within a procedural application program like Ada. Chastek et al. warns against using this technology in the development of Ada applications since it in effect creates a new language that "...no longer conforms with the Ada Standard." [14:3]. A solution to creating an Ada to SQL binding, or an interface independent of pre-compiler technology, was established by the work accomplished in [14; 13; 17; 16]. The model developed by Graham et al. [14; 13; 17; 16] is called the Structured Query Language Ada Module Extensions (SAME) and will be discussed in detail in Chapter IV. A solution design using the requirements of the STEMdB will also be presented in that chapter.

Top Level Requirements Change. A significant area in the STEMdB design that was not addressed by the requirements was the identification of end users of the STEMdB tool itself. Due to the widespread need for this type of tool in both industry and the government, and due to the limited resources of any DOD organization, the tool should be developed with remote usage users and multiple platform configurations in mind. Building on the concept of abstract interfacing and hardware independence, extending the STEMdB requirements to support remote users and multiple platform configurations becomes trivial. The requirements should reflect this as a design goal. Given that this is a valid requirement, the following implications must be addressed by the design:

1. With the many variables associated with CASE tool selection, the tool should be built and tailored as a decision support system. It should be built so that a user who has no previous knowledge of evaluation and selection processes can use it effectively. Decision support systems do not have the expert knowledge that an expert system has, but they do have some degree of domain knowledge which allows them to provide the user with an informed decision. With this in mind and armed with the knowledge of information proliferation from Chapter I. any tool that helps reduce volumes of information into some useable form benefits all users. This can be understood by looking at the only alternative to such a tool which would be a user inept at evaluation and selection processes attempting to select an appropriate CASE tool. Such a user more than likely would accomplish an ad hoc evaluation and selection with limited information that may not meet his/her needs.

2. Incremental development would allow various phases to the tool to be developed to meet the goal of remote usage with multiple platform configuration. The first phase would involve strictly in-house use at the STSC, the second phase would involve the STSC sending personnel out to remote locations (locations other than the STSC) to both

test the remote database access and the remote application operation. Remote testing could also involve a joint training session with the remote organization on how to operate the STEMdB tool. The next phase would allow pilot remote organizations trained in the correct usage of STEMdB to operate it remotely and supply feedback. These first three phases would have to be repeated for all target platform configurations. The final phase for any configuration would involve providing both the tool and tool training to any organization that has a need and that has the remote interfaces, software and hardware to support the tool.

One of the major problems associated with this type of approach is the complexity that results from relationships between different hardware platforms, different user interfaces, and different database interfaces. This complexity can be reduced by designing the system around abstract interfaces to these areas. The STEMdB itself would have to be configured by using "concrete interface" [17:6] modules in the implementation part of an abstract interface. Concrete interface modules, according to Graham, are the modules that are implementation dependent. The use of concrete interface modules along with their enclosing abstract interfaces, would allow an application such as the STEMdB to be portable across different configurations. As the need for more remote platform support becomes necessary, the STSC would have the option of extending the STEMdB's target platform domain to meet this need. In addition, the STSC could provide to the Ada software development community a library of these interfaces. The Ada development community would then have the capability of making their Ada applications more portable across different configurations.

The one interface in this section that may not be feasibly abstracted is the user interface. This may be the case if the STEMdB user interface is built using a commercially developed user interface application. One example of where a commercially developed user interface could not be feasibly abstracted out of an application is a HyperCard program running on a Macintosh computer. This was discovered by Lawlis while implementing her ASSIST prototype in Hypercard.

3. Other design changes resulting from remote usage would require that the STEMdB:

a. Provide an initialization routine in the abstract application interface that creates the tables that will mimic the database format of the STSC database.

b. Provide capability to download and import the data into a users local database. This capability could initially be provided as a manual procedure. In this case, users must understand how to import data based on their own database and the format of the downloaded data (i.e. comma separated text, tab separated text).

c. Add the capability for the tool to check downloaded data against a date flag. By ensuring that upon "download date expiration" any further selection work will not be allowed to proceed unless new information is downloaded, the STEMdB will help maintain the data currency requirements. Due to consistency considerations that local database information, additional capabilities will need to be provided for local database currency updates. For instance, the local weight set data will be updated with any new default weight set data in the currency update process. If steps are not taken to preserve the original user defined weight set data, it could be overwritten in the update process. Similarly, the remote tool will need to be sure not to upload any Selection_Set information (since this information only makes sense when the remote user identifies tool sets important to his/her organization) and it will have to provide for restoring software_char_weights. Two constraints that must be checked on the newly uploaded data would be that all restored linkST Tool_ID entries have matching tools in The_Tool and that all restored GCS_ID's in software_char_weight have matching GCS_ID's in the General Software Characteristic table. These consistency checks are necessary since the newly uploaded data may have added or deleted information that the earlier system used. For instance, a tool could have been deleted from the database.

d. Finally, the remote tool should be able to check an application integrity constraint that would guarantee that the remote application and the STSC database are compatible. This would ensure that there were no structural revisions to the database design after the STEMdB remote application was released.

Improvements to Be Made Within Context of Present Design. This section will suggest improvements to the Data model, the Formulator functionality, the Evaluator functionality, and the Selector functionality.

Data Model. There were six areas that needed to be addressed further within the Data Model.

The first area concerns the relational database design goals of Boyce-Codd normal form/Third normal form(**BCNF/3NF**), lossless join decompositions and dependency preservation. According to [23:209] these design goals are principle criteria for good relational database design. The overall minimal requirement in any database design is to reduce update redundancies while preserving functional dependencies. Either normal form accomplishes the goal of reducing update redundancies with BCNF also minimizing those redundancies. The difference between BCNF and 3NF is that 3NF is a less restrictive form (it can have some redundancy) that maintains functional dependency preservation whereas the BCNF form is a more restrictive form that guarantees minimal redundancy at the possible cost of dependency preservation. A definition of BCNF form is provided in the following analysis sections. Since 3NF form was not needed in the following analysis sections, its definition is not discussed. Dependency preservation means that after a database scheme is decomposed into its sub schemes, each of the problem space dependencies "can be tested in at least one relation in the decomposition"

[23:182]. According to [23:181], when accomplishing relational decomposition, designers must ensure lossless decomposition/joins. This means that the designer decomposes a scheme so that no functional dependencies are lost as a result of the decomposition [23:184]. There was no indication in the STEMdB documentation of requiring any of the design goals discussed in this paragraph.

Without knowledge of how the STEMdB relational data model was designed, more confidence in the design had to be gained. To gain this confidence, a canonical cover with dependencies defined in Table 1, was derived. A canonical cover is a minimal set of functional dependencies that fully defines a schema with minimal repetition of information. A canonical cover of dependencies must be able to completely derive all original dependencies without adding any additional information. The form of each component functional dependency in the cover is required to have a unique left hand side. Table 1 lists one possible cover that fully defines the STEMdB Schema [5]. The syntax $\mathbf{T} \Rightarrow \mathbf{A}$ is read "**T** implies **A**".

For better readability of candidate key results, and for functional dependency processing derivation, it is desirable to represent keys in some abbreviated form. The acronyms used in Table 1 represent the keys/attributes of Figure 6. In general, if a list of attributes was defined by long words in Figure 6, for instance the software_char_score non-key attributes, then the acronym used to describe this list was written with distinguishing capital letters followed by an "_a". One example of an acronym conversion follows: The software_char_score non-key attributes in long hand would be written "function_score, quality_score", but this new notation reduces to the following notation: SCS_a. When a key attribute was represented in shorthand, it was just distinguished by an obvious set of capital letters. Table 2 maps the acronyms of Table 1 to the keys/attributes they represent in Figure 6.

Table 1. Functional Dependencies of STEMdB Schema

GSC, T	$\Rightarrow$	SSC_a, Parent_GSC
WS, GSC	$\Rightarrow$	SCW_a
A	$\Rightarrow$	GSC
WS, T, GSC	$\Rightarrow$	SCS_a, TS_a
EVAL	$\Rightarrow$	date
WS	$\Rightarrow$	default
GSC	$\Rightarrow$	GSC_a, A
T	$\Rightarrow$	T_a
QUAL	$\Rightarrow$	QUAL_a

The main result of the derivation of a canonical cover was the identification of a STEMdB functional closure. Closure analysis identified {T, A, WS, EVAL, QUAL} or {T, GSC, WS, EVAL, QUAL} as the candidate keys. Candidate keys provide the minimum functional information a database schema must have to fully identify all functional dependencies. This analysis shows that given an Area or GSC, a Weight Set, a Tool and evaluator/quality information all other dependencies in the database can be derived. The candidate key with the GSC in it will only work, however, if that GSC is a root node in the present design.

Upon further analysis each relation scheme of the STEMdB, in the original Figure 6, turns out to be in a BCNF form that is dependency preserving. This is true since all but two relations were of the form of Super Key $\Rightarrow$ attributes. To check for dependency preservation, any relation that has functional dependencies beyond those of Super key $\Rightarrow$ attributes must have these dependencies preserved and accounted for in the cover analysis. The two relations that had additional constraints were the root_node and the linkGG. Additional constraints surface whenever there is a one to one or a one to many relationship. The two additional functional dependencies resulting from the one to one relationship between The_Area and the General Software Characteristic were incorporated into Table 1 and were maintained in the design by the cardinality. The same can be said for the child GSC_ID determining its Parent GCS_ID, which results from the one to many relationship in linkGG. Since all relations that make up the STEMdB are in dependency preserving BCNF form, the STEMdB Schema is in BCNF. These results allowed this researcher to conclude that the schema now reflected the database design goals established earlier.

During the course of this analysis a problem was discovered in the context of the problem space, however. Ambiguities were resulting from the design using two binary relationships to define the tool_score relationship. This relationship is truly a three way relationship that depends on a root GSC node, a tool and a weight set. By designing it as a binary relationship the integrity of the data was compromised. The best way to show this is through a scenario that uses the structure of Figure 6.

1. Given a tool, T1, that maps to two evaluation domains, A1 and A2 through two root GSC nodes, G1 and G2.
2. Given one weight set, WS1, that maps to **both** G1 and G2.
3. Provide the tool_score for T1 using WS1.

As the reader can surmise there are actually two different domain dependent scores that the tool can be assigned based on whether A1 or A2 is selected. This ambiguity can be eliminated from the design by specifying a GSC that defines which domain the score is desired in. The original STSC design omitted the key attribute of GSC_ID. The redesign in Figure 8 implements this correction by connecting the tool_score relation to the Software Char Weight aggregate thus inheriting the GSC_ID key from it.

Table 2. Map of Table 1's Relationship to Attributes

T	=	TOOL_ID
A	=	AREA_ID
EVAL	=	EVAL_ID
QUAL	=	QUAL_ID
GSC	=	GSC_ID
T_a	=	The_Tool non-key attributes
GSC_a	=	General Software Characteristic non-key attributes
WS	=	WEIGHT_SET_NAME
SCW_a	=	software_char_weight non-key attributes
SCS_a	=	software_char_score non-key attributes
TS_a	=	tool_score non-key attributes
EVAL_a	=	The_Evaluator non-key attributes
Parent_GSC	=	Parent_GSC_ID
QUAL_a	=	The_Quality non-key attributes

The second area that needed to be addressed further within the Data Model concerns table optimizations that can be made in the design. The original STSC design lacked any reference to a relationship called root_node between the The_Area and the General_Software_Characteristic entities. The design of Figure 5 references the theSEATree as a link in the The_Area table and it references the keys of the The_Area table as a key in the General_Soft_Characteristic table, yet the rest of the design never references how these are connected. It appears as though the system designers were incorporating an optimized design into the requirements document while omitting the source design. According to Dr. Roth in [6], it is important to keep a record of the original design prior to any implementation optimizations. In order to re-host the design on a different target architecture (on a distributed system for instance) at some future date, this record along with a record of optimizations will help to prevent any reverse engineering from being required.

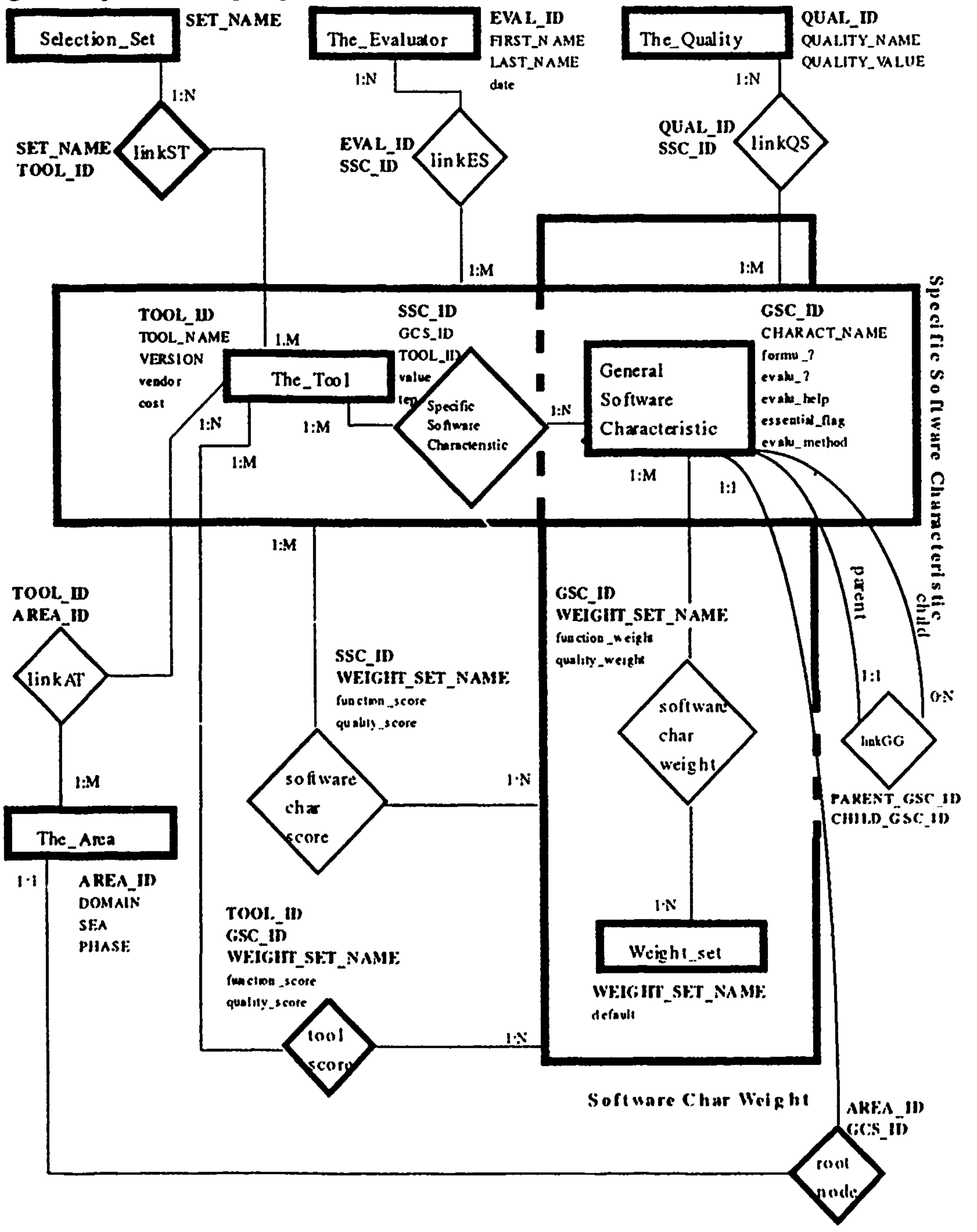

Figure 8. Entity Relationship Diagram Revised

By incorporating a root_node linking relationship (as shown in both Figures 6 and 8) the connection between the General_Software_Characteristi: and the The_Area tables can be established and any previous optimization is eliminated. Since the basis for the previous optimization was not stated, it's existence could not be justified. The one to one nature of the new root_node relationship implies that table optimization can be accomplished by eliminating the root_node relationship table and by adding keys to the appropriate tables. Since The_Area requires exactly one root GSC node, the GSC_ID of the root node can be linked to exactly one entry of the The_Area table by adding it as an attribute. Since a GSC does not necessarily have to be linked to an entry in the The_Area table, adding an AREA_ID key to the General Software Characteristic table would be a waste of storage space for the design in this chapter. Consider, for instance, that only one node of all GSC's for an area must have an AREA_ID attached to it. All other GSC's have an implied association with that area through their linkage to the root node. Therefore by adding the attribute of root_nodeID to the The_Area table and eliminating the need for the root_node relation, functional dependency between the two entities is maintained while storage space is saved. The original design which associated an area with every GSC was wasteful unless the developers had a design goal that was not specified in the requirements document. By eliminating the linkGG table and placing a parent_GSC_ID key in the GSC table, another table optimization can be established. This would be functionality correct since every GSC has exactly one parent, except for the root node GSC which has a null parent.

Another interesting twist to the old design can be seen by concentrating on the hierarchy of GSC's. Early in the research a STEMdB designer commented that the GSC structure was more of a directed graph than a tree since it could have more than one parent assigned to the same child. Allowing this type of design would save space with common GSC's shared by different description trees. Since the original design had an

Area associated with every GSC node, there would have to be a way of coding multiple Areas into each GSC node. Associating multiple areas with one GSC node would change the cardinality of the original design to many to many and would require that another relationship table be added to the original relational design of Figure 5. During this research, the multiple parent concept was discarded by the STEMdB designers. The complications associated with this type of design were determined to out-weigh any of the benefits. From one perspective, this could be considered a good design tradeoff decision since it does reduce the complexity of the design. From the big picture perspective, this design decision may have placed unnecessary restrictions on the design too early. Chapter V will present a more detailed argument against early restrictive design decisions.

The third Data Model area requiring re-work concerns the naming conventions used in [19:26-28]. The relationships Tool_Score, The_Score and The_Weight are not listed in the linking relations section of the design document. These three relations are linking relations as can be seen by their counterparts of Figure 6, therefore they should be identified as such.

The fourth area has to do with optimization questions about the Specific Software Characteristic table. The Specific Software Characteristic entity can be thought of conceptually as consisting of itself, The_Evaluator and The_Quality entities. The present design requires table joins to be accomplished between the Specific Software Characteristic and The_Evaluator and the Specific Software Characteristic and The_Quality tables through the linking tables of each. Join operations are time expensive when one considers that the underlying operation depends on searching for matching attributes in the cross product of two table's entries. To access all SSC data on any area, the access time will be of order (Number of Specific Software Characteristics * Join time required per characteristic fetch). This join time is functionally dependent on the size of the Specific Software Characteristic table (maximum = 1000 entries), the Evaluator table, the Quality table, their linking tables, and the efficiency of the search algorithm. If memory space is not a limitation, the lag time associated with the present design can be eliminated by placing all of the attributes/keys into one table, the Specific Software Characteristic table. A drawback of this is the limitations that result from a fixed design of the number of allowable entries of evaluators and quality values per characteristic.

The fifth area concerns the re-design of the conceptual relationships of the Selection_Set and the Weight_Set.

Selection_Set Re-design. Presently the Selection_Set retains only a list of tools that were evaluated under some domain and some weight set. To remind the selector of the original context of the selection set, an automatic connection to both an area and a weight set should be associated with all selection sets. To address this design in the data model, a four way relationship link could be created connecting the Selection_Set, The_Tool, The_Area and the Weight_Set. Figure 9 shows what this would involve for the entities discussed. The linkSAWT relation would then replace the linkST relation of Figure 8.

Weight_Set Re-design. For more flexibility of the design, there should be three levels of default weight sets: the automatic default that assigns all nodes within the same level equal weights, an explicit Formulator system default that assigns weights to nodes based on empirical evidence for a domain, and the user definable default. This design could be implemented with an optional one to one relationship between the The_Area and the Weight_Set tables or it could be implemented with access time optimization by adding the attribute of system_default_weight_set_name to The_Area table. One other way of implementing this would be to add an "empirical_weight" attribute to the GSC Table. This re-design would require that the Formulator be re-designed to create and store the default weights. A further justification for this re-design will be presented in the analysis section on Formulator functionality.

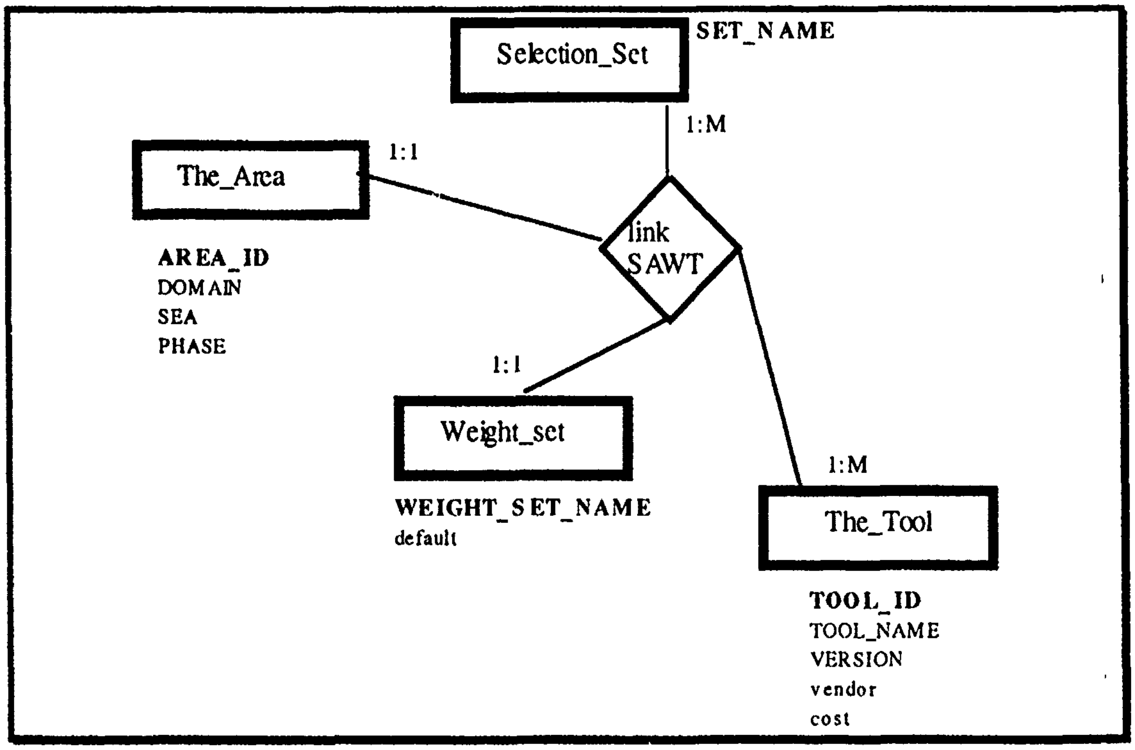

Figure 9. Selection_Set Redesign

The sixth area concerns a more extensive conceptual description of the data design. Although a section on referential integrity was provided in the STEMdB requirements and design document, a conceptual discussion of the entities was not provided. A conceptual discussion provides a better word picture of how entities relate to one another and how they affect one another upon update/deletion. The following presents a suggested solution to this problem:

Conceptual Description of the Data Model. The conceptual relationships between the different entities in Figure 6 can be understood by considering the optionalites and cardinalities as follows:

1. The_Tool - Has a mandatory relationship with both the Selection_Set and the SSC. This implies that eliminating a tool from the database would cause records in the Selection_Set table to be deleted if the tool was a member of that selection set. The same can be said for the connection with the SSC table. If a tool was deleted then characteristic values evaluated for the tool would no longer make sense to the STEMdB model. Eliminating a tool would not affect the integrity of the Weight_Set or The_Area entities, however.

There can be many tools associated with many areas or domains, selection sets and weight sets. Only one tool can be associated with many different SSC's.

2. The_Area - Eliminating an area/domain would affect the integrity of the The_Tool entity, only if the area eliminated was the only area that a tool applied to. A domain elimination also affects the integrity of the GSC table, since there is a one to one mapping through the root_node relation. This would cause an entire description tree with a GSC as its root node to disappear, which would result in all related SSC's to disappear.

There can be many domains for many tools but there is exactly one domain for one root_node GSC.

3. General_Software_Characteristic - The GSC has a mandatory relationship with all related entities except for its 'child' recursive relation. Therefore the integrities of all related entities, including itself, could be affected by an elimination of an instantiation of the GSC.

4. Specific_Software_Characteristic - This entity has mandatory relationships with The_Evaluator and The_Qualities. This makes sense since these two entities can be considered a part of all unique SSC's.

5. Weight_set - Has a mandatory relationship with the GSC. Many weight sets can be associated with many tools, SSC's and GSC's.

6. Selection_Set - This entity does not have to exist with respect to a tool. There can be many selection sets associated with many tools.

7. The_Quality - The_Quality values must exist for the SSC. Many different quality records can be associated with one SSC. This makes sense in the context of an evaluation since quality values of one characteristic are associated only with that characteristic.

8. The_Evaluator - The_Evaluator must exist for the SSC.

This completes the conceptual discussion of the data model as well as the other data model discussions. The next three sections present a discussion of functionality provided by the three sub-tools, the Formulator, the Evaluator and the Selector.

Formulator Functionality. One improvement to the functionality of the Formulator which could have significant impact on the STEMdB tool utilization, would be to add an empirical weight set insert capability. Providing an empirically defined default weight set, that has a solid statistical basis for a given CASE tool Software Area of Interest (SAI), has the potential to make the STEMdB Selection process more automated. It would become more automated by allowing the user to choose an area then sit back and wait for the resulting tool list suggestions. The more automated a system is, the simpler it is to operate and the more usage it will get provided its results are valid. In order to get this statistical basis, an internal monitoring routine could be implemented to record weight set information identified in a STEMdB session. The system could also provide an automatic reporting routine that the user would use to send the raw data to the STEMdB developing organization. This reporting capability addresses the feedback requirement that an evaluation and selection system process must have to continue to improve [1:7]. Once this capability is added and good statistical bases for certain SAI's are developed, the next natural step would be to add a default weight attribute to the General Software Characteristic table. By adding this attribute the initial setup of searching the software_char_weight table for default weights could be avoided. If these empirically defined weights do turn out to be used the most often, then adding this attribute reduces total processing time.

The Formulator documentation is ambiguous about the build process and the hierarchy of characteristics. Specifically, it does not describe what types of characteristics are and are not allowed to structurally follow at a lower level in the build process. The STEMdB has five node types that can be assigned at any level in the build. Incorrect hierarchical ordering of the types could nullify functionality that exists in a node's children. For instance, if a single item checklist node is the parent of a node that is an evaluate children type, and if the single item check node is evaluated, then the functionality of the parent is assigned a one and its children do not add or detract from this functionality. This result is not consistent with the requirements that the functionality of the parent be determined by its children. The Formulator should provide build restrictions on structures that are illegal and these restrictions should be documented.

Evaluator Functionality. More flexibility should be provided to allow for multiple evaluations of functions/qualities. The STEMdB design concentrates on a final evaluation result for a function/quality that is statistically arrived at outside of the scope of the STEMdB. The Lawlis [24:315-316] design accomplishes this task inside of her tool by incorporating a "Summary" package as part of the Knowledge Base. Part of the functionality of her Knowledge Base is to statistically average different evaluations of an implementation and store the results of this process in a Summary package. Both Lawlis and the evaluation and selection team of [1] emphasize the need for multiple evaluators of subjective areas.

The design limitation of twelve quality attributes associated with every characteristic is too restrictive and should be lifted. This researcher could find no justification for enforcing this limit, other than a user interface that was tightly coupled with the design. This restriction limits the STEMdB's capability for expansion. The Knowledge Base discussed in Chapter IV will provide the evaluator with the qualities associated with the STEMdB at any given time. The concept of the selector viewing these qualities can also be addressed by calls to the Knowledge Base.

Selector Functionality. The first and most obvious area in the Selector functionality that needs more work is the process of identifying and weighting important tool characteristics. The ASSIST tool accomplishes this process by only processing the features and criteria that the user identifies and assigns relative importance to. The STEMdB tool assumes that all nodes will be visited. Those nodes that are not visited and whose parents are not assigned a weight of zero, receive the system default weight and are still used in the selection process of arriving at a score. This method places the unnecessary restriction on the user of processing a minimum number of characteristics in order to ensure that only his/her specific requirements are addressed. For instance, if an SAI tree had five top levels of functionality, and the user only wanted the tool to identify candidate tools based on exactly two of these areas, then the user would have to visit each of the three undesirable nodes just to assign them a zero weight. The simplest solution to this problem is to provide the user with a resource that equates to assigning zero weights to all unprocessed nodes. This resource could also be provided at a higher level of abstraction where the user can toggle it on or off. When on it would monitor all nodes processed and then zero out all that had not been processed just prior to initiating the scoring process.

Another area requiring more work is closely related to the one just discussed and has to do with marking a characteristic as essential in the selection process. The Lawlis design required that all nodes identified in the process be evaluated favorably for a tool to be scored at all. The STEMdB requires that all nodes marked as essential by the Formulator be evaluated favorably to be scored. It also provides the user the option of marking each characteristic as essential but this option is provided outside of the process of assigning weights. In effect for the user to tailor the system to his/her needs, she/he must go through an additional process of marking essential characteristics while being forced to accept system defined essential characteristics. The fact that a user identifies a characteristic and assigns a weight to it, strongly correlates to that characteristics being desirable and possibly essential. The STEMdB should provide a resource that when toggled on, identifies all user visited and accepted characteristics as essential. In addition, a resource to remove the Formulator defined essential constraints from the process should be provided.

Since using a linear weighting approach to multiple criteria can result in decisions that the user may not want (and in the STEMdB design the user may not know that he/she has been provided with a decision that doesn't meet user criteria), the designers should state explicitly in the design documentation how these misconceptions and bad decisions will be avoided. This problem with a linear weighted approach to multiple criteria DSS systems is well documented.

Another area requiring re-work is the user interface presentation of data. Both too much information and unnecessary information are provided to the user. The concepts of leveling of information and of allowing for different user experience levels are not addressed in the STEMdB design. For instance, the user is provided with a window that shows the characteristic names structured within the form of an indented, numbered, scrollable list. The novice user does not need to know how the characteristics are structured and he/she doesn't have to initially have access to the complete list of characteristics. After working with the ASSIST prototype from the Lawlis dissertation and after analyzing the STEMdB design the following conclusions were made. Abstraction out of a multi-level tree view and into a default list view at any given level will provide the novice user with as much information as needed. A pop-up menu containing the remaining characteristics at a given level can be displayed by the selector when additional characteristics at a level are desired. The Formulator-defined empirical weight could be used to indicate which characteristics should be presented at which level. A user experience level function can be designed to provide more information to the user based on the experience level chosen.

Another problem discovered in the Selector processing results from an implicit rather than explicit way of handling tool cut-off threshold processing. Although the STEMdB design implicitly allows the user to assign a threshold level by allowing the user to require that the root node be above a certain score, it does not d[illegible]s explicitly. The design also made no mention of setting a default cutoff threshold for the tool score process. Both Lawlis and an IEEE group in [24; 1] specifically identify that the user must be allowed to modify cutoff thresholds. Both references also advocate providing a system default cutoff threshold. With this justification, the STEMdB should provide a system default cutoff threshold and it should explicitly give the user the capability to change this default.

Another unnecessary functional restriction in the Selector results from presenting the user with two disjoint scores, one for the functionality and one for the quality. The STEMdB could easily present a combined score to the user. Adding this capability forces the tool ranking mechanism to be altered. A single combined top level ranking of scores (based on user defined weights) should become the default ranking mechanism, while existing ranking methods should be made available at another lower level of detail.

The next area that the designers of the STEMdB were negligent in addressing was the concept of evaluations existing in different states of completeness within the database. Although the concept of returning to marked areas to continue an evaluation was addressed, the concept of allowing selections on these partially evaluated tools was not addressed. Lawlis in [24] emphasized the importance of making sure that evaluations are done based on the same criteria. If some GSC is not evaluated in one tool but is in another, would the STEMdB tool still compare the two tools and score them? Since partial evaluations will be allowed, there needs to be some mechanism for tracking when a tool is completely evaluated within the context of an area. This mechanism must be accessible to the Selection subsystem. The STEMdB designers could design for two levels of completeness in a tool evaluation: a degraded level and a complete level. With software updates and improvements happening so rapidly, the degraded level could represent a set of characteristics that is empirically proven to support most minimum user selection criteria. This type of leveling would allow quick degraded evaluations of new and upgraded tools to be accomplished in minimal time. This would also address the issue of maintaining currency in the evaluation database. Another solution to this problem is addressed in the design of ASSIST [24]. This design recognized that evaluation information could be incomplete between different tools. Instead of forcing the user to accept a degraded selection suggestion, it allowed for incomplete evaluations by summarizing this incomplete information in a selection report. All candidate tools were scored based on the same process, even if some were missing information on non-essential criteria. Users could then judge whether to accept the output or revise their own inputs based on the reported missing information. These selection reports should be provided so that they supply the user with the context of how a decision was arrived at.

They should provide a list by tool of characteristic assessments that were missing, a list of tools that did not make the system threshold cutoff, a list of tools that did make the cutoffs, and a list of tools that were not considered because they did not have an essential characteristic [24:103]. In [24:73-103], Lawlis provides a more thorough discussion of how Selector or decision support logic should be defined.

Another area in Selector processing that was overlooked was providing the user with the capability of choosing evaluation information based on evaluator "type". By providing a "type" attribute associated with an evaluator, a user can identify a group of evaluator types whose information is most valued. For instance, a non-technical purchaser on a limited budget desiring to purchase a personal computer would be more interested in evaluations of other non-technical users, since technical users tend to have more sophisticated requirements. Providing for selections based on evaluator type creates another mechanism for narrowing down the candidate tools of a certain SAI. Providing this mechanism will also help to address the area of design sensitivity to user requirements.

Summary

This chapter provided a description and analysis of a CASE tool evaluation and selection tool. It compared the ongoing prototype effort of the STEMdB against goals established in the literature. It used another prototype called the ASSIST as a basis for comparison since both tools were developed towards the same goals, although the ASSIST addressed these goals in a more abstract manner.

Two significant design omissions or unnecessary restrictions were discovered in both the design methodology and in the system's ability to communicate with databases. Lack of a specific design methodology to follow and the Object Oriented problem space suggests that the design be approached with an Object Oriented methodology. This researcher also identified the need to use Ada as the implementation language. Ada facilitates abstract interface designs as well as designs using good software engineering principles and thus using Ada has the potential to make the system more maintainable, upgradeable and portable.

Several present design re-work areas were identified by this researcher from the data model and functional model perspectives. The initial design was erroneous in its design of the tool_score relationship, and table optimizations could be made. Also the functionality of the Formulator, the Evaluator and the Selector could be improved by the eliminating the deficiencies identified in this chapter.

IV. New Approaches to STEMdB Design

Overview

This chapter will present two top level designs that address the two new approaches identified in the STEMdB analysis of Chapter III. It begins by showing how the object oriented design of the ASSIST (Figure 1 in Chapter II) maps to the parts of the STEMdB design (Figure 4 in Chapter III). Using the ASSIST framework in the context of a STEMdB system, it then presents a top level STEMdB object oriented design. The emphasis in this top level design is placed on the structure and methods of interaction with the data store. In particular, this top level design emphasizes the Knowledge Base subsystem's role in this interaction. The chapter then concludes by describing and applying a new approach towards database interface design, the Structured Query Language (SQL) Ada Module Extensions (SAME). This method will be used in establishing the top level interface design for the abstract system interface that will allow the STEMdB to communicate with SQL databases.

Top Level Design of an Object Oriented STEMdB

For the reasons stated in Chapter III the design of the STEMdB can be improved in two ways, by redesigning the structure and dependencies to reflect a knowledge base object oriented design and by abstracting out the interface to the database. A re-design into an ASSIST-like object oriented framework is presented in Figure 10. The differences between this design and the original ASSIST design result from the addition of the Formulator subsystem and from the renaming of the Knowledge Acquisition subsystem to "Evaluator" and the Decision Support subsystem to "Selector". One other difference results from the ASSIST maintaining the database inside of the Knowledge Base Subsystem. The new design consists of a data store, three procedural subsystems, and two interface packages. Communications between the objects in this design are shown by the directed arrows. Before proceeding with a more detailed description of this design, the merits of the new design versus the old are presented.

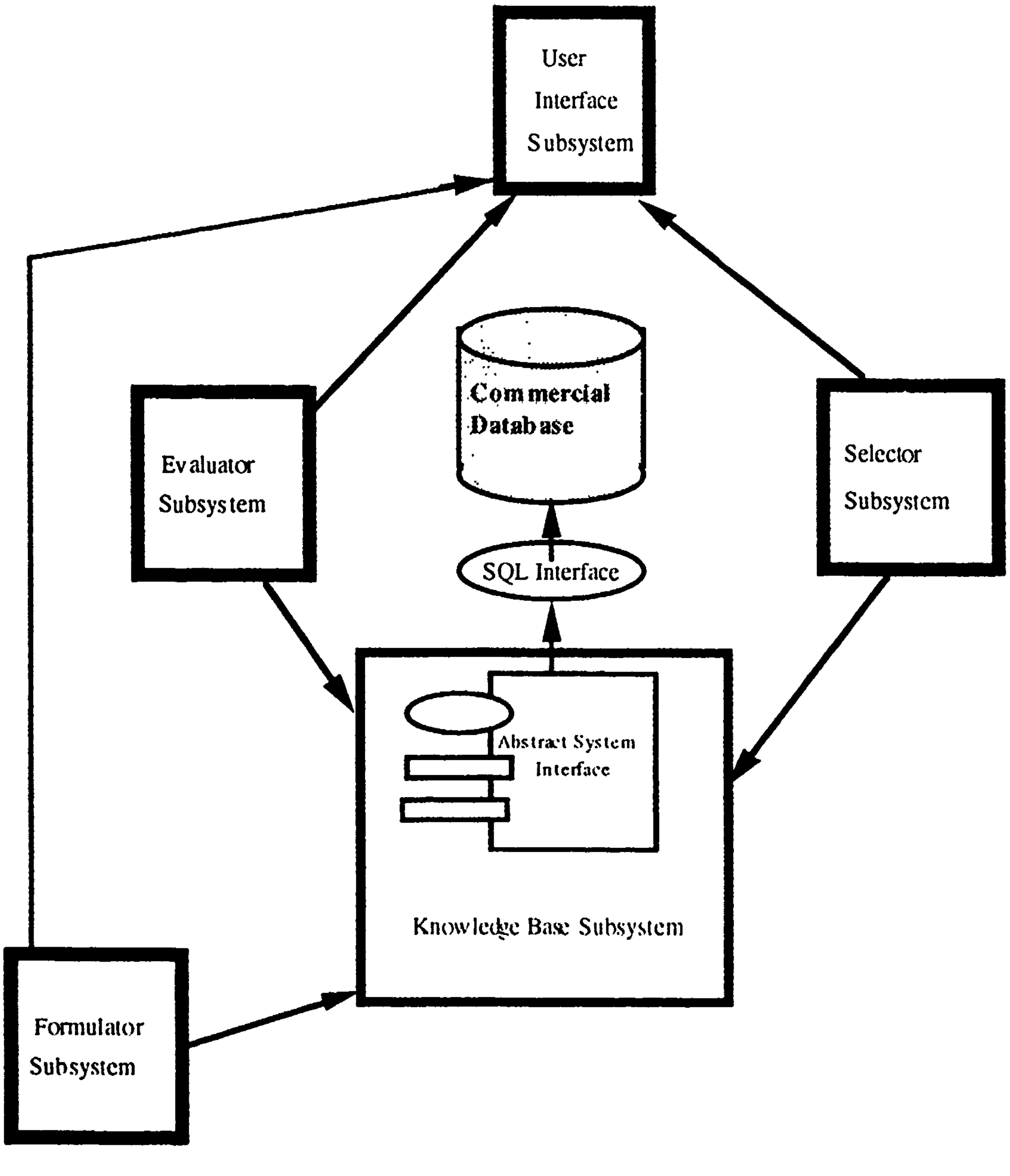

Figure 10. Object Oriented STEMdB Design

A breakdown of the acronym of the original design, "stemDB", shows how the designers of this system placed more emphasis on the database, "DB", than the Software Tool Evaluation Model, the "stem". Consultations with the original designers also confirmed this conclusion. To understand how the new design emphasizes a knowledge base, and therefore the Tool Evaluation Model, versus the original database emphasis, Figure 10 can be compared to Figure 4 and Figure 7 (both of these figures are repeated on the next page for ease of reference). From Figure 7 and the design description of the Front-End module, it could be inferred that there is a tight coupling between the Front-End module and the Database engine. Further, since there is no discussion of de-coupling the Front-End from the three subsystems in Figure 4, the design allows by omission the possibility of a tight coupling between the three subsystems and the data structures maintained in the Front-End. Tight coupling among separate modules in any software design causes the design to be less maintainable and more prone to errors due to the dependencies between modules. Tight coupling also supports an environment where system state information can be de-localized and spread throughout the design. The Knowledge Base object oriented approach of Figure 10 eliminates the intra-module coupling by using well defined interfaces which provide methods and types to calling modules. The methods or operations are suggested by the rectangles extending from the subsystem box and the types are suggested by the extended ovals. The arrows in the figure represent communication between modules. For instance, since the Evaluator and Selector have no knowledge of the information contained in the Database, both the Software Area of Interest (SAI) identification and characteristic identification methods must be requested of the Knowledge Base before either process can proceed. With this comparison completed this chapter will now discuss the design in more detail.

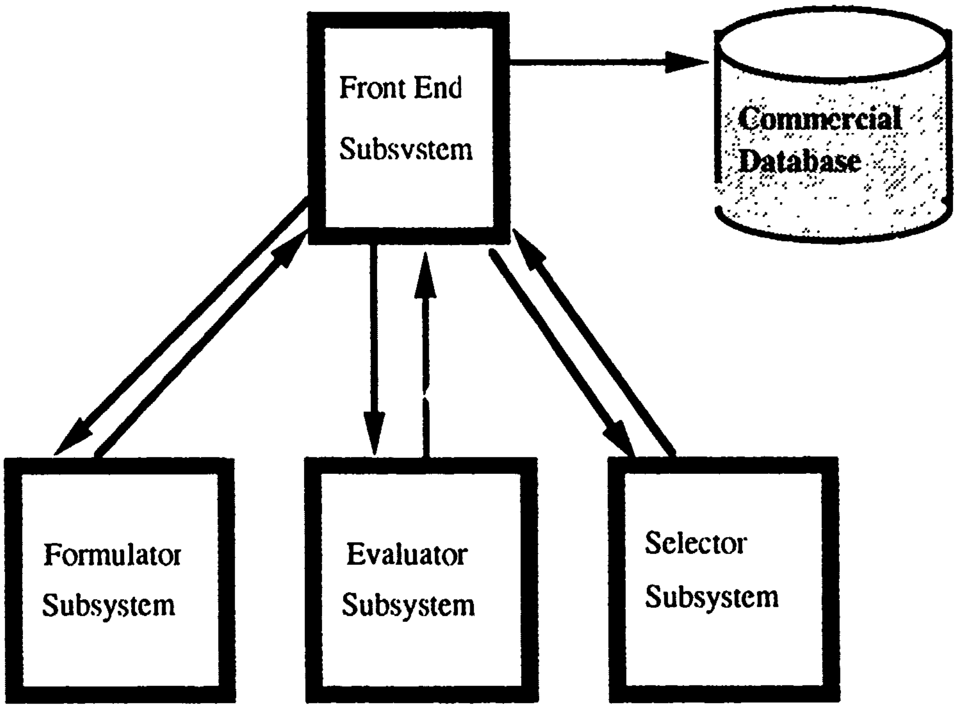

Figure 4. Original Design Components (repeated from Chap 3)

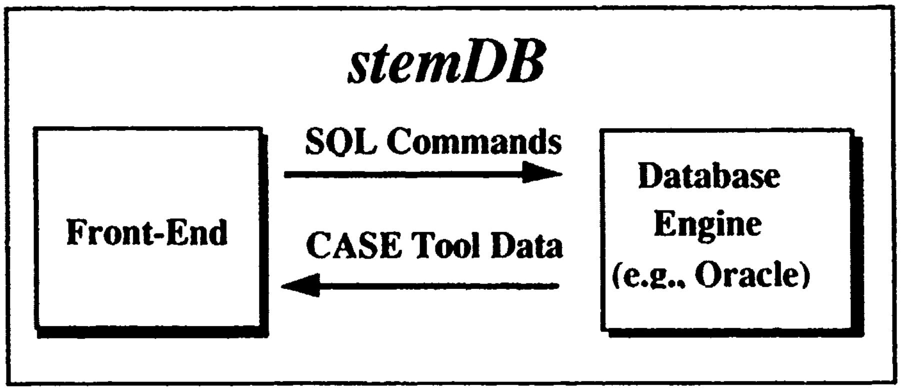

Figure 7. STEMdB Basic Components (repeated from Chap 3)

The processing of this subsystem and other subsystems will be described in the following sections using terms that are more generic than the terms of the original STEMdB design. Specifically, the term "semantic net" and "knowledge frame data" will be substituted for the terms "description tree" and "node data", respectively. They will also be used interchangeable with the terms "structure" and "characteristic data", respectively. The justification behind this switch in terms is centered around designing at a high level of abstraction. The use of the terms "tree" and "node" forced the "implementation" design decision that the data would be organized in a tree structure too early. When implementation decisions are bound to the design too early, they limit the design and cause the system to be less robust and less maintainable by tempting designers to tailor the design for a designated structure. Specifically, the original STEMdB design was limited by binding the system data structure to a tree structure. One limitation that resulted from this decision can be seen by realizing that the tree structure eliminated the possibility for reuse of common characteristics by not allowing multiple parents. By eliminating this possibility the design was not developed in a more general sense. Had it been designed in the more general sense, then binding to a single or multiple parent structure would ideally occur at or close to implementation time. By forcing this decision at the end of the design, either a tree structure or a directed graph could be implemented.

Design Discussion of New Formulator. The Formulator Subsystem conceptually exhibits the behavior as described in the requirements document (with the improved behavior identified in the analysis section superseding weaker behaviors) and it achieves this behavior using an object oriented methodology. Figure 11 provides a more detailed view of the internal design of the new Formulator. It consists of a Formulator Build processor which requires the resources provided by the four resource packages shown. Resources are simply methods or operations and object types. The shaded packages are resources tailored to the Formulator Build Process. Clear packages have some resources unique to the Formulator Process, but also have resources that are common across all three subsystem processes, the Evaluator, Selector, and Formulator. There is the possibility that the Present State resource packages across all three systems (see also Figures 12 and 13) could share resources but this would have to be addressed in a more detailed design. This possibility becomes stronger when one considers the Evaluator and Selector processors and their dedicated packages. For instance, both of these processes will desire state information about characteristics that were visited and altered.

There is no implied ordering associated with the internal sub-processes other than initial setup requirements which occur in the top two procedure boxes of the Formulator and Selector, and the top three of the Evaluator.

To understand how this processor accomplishes its job, the following Top-level Structured English describes the processing that the new Formulator must accomplish. This section and all following processing sections specify in their comment areas (which are preceded by two dashes) how the Knowledge Base (KB) subsystem interface aids in their processing.

```
Initialize to Formulator_start_state;
   -- Call KB routine "Initialize_KB_formulator" through Present State of Build
   --       Resources initialization method.

Provide toggle capability to Set_user_level;
   -- Call KB methods "set_expert" or "set_novice" through initialization resources.

Provide ability to open new or old Software Area of Interest (SAI);
   -- Call KB methods "define_SAI" or "update_SAI".


Provide ability to mark a description_area as released/not-released;
   -- Call Build resources "release" or "imature_area" which will then call the
   --       KB routines.
```

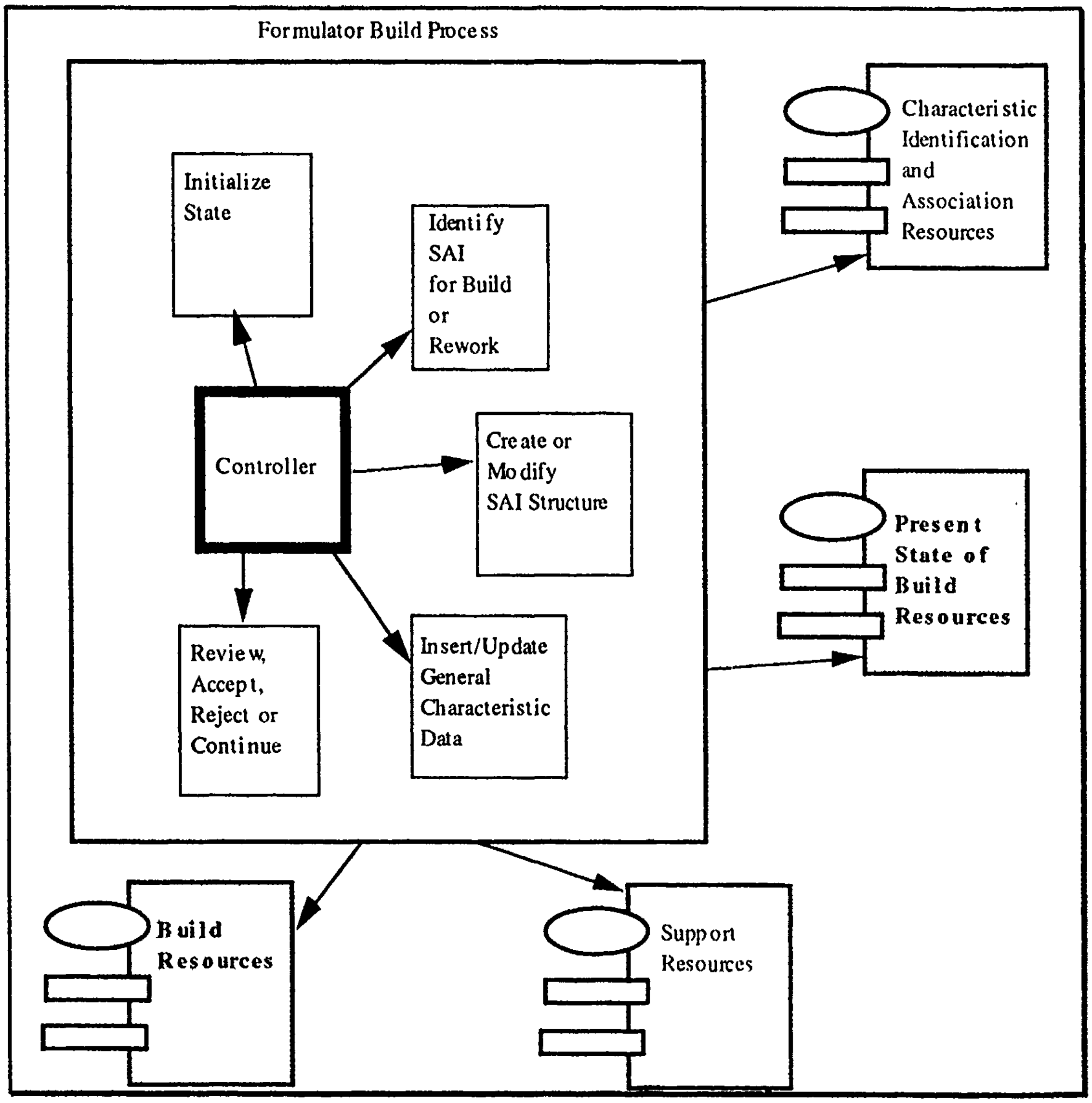
Formulator Build Process
Initialize State
Identify SAI for Build or Rework
Controller
Create or Modify SAI Structure
Review, Accept, Reject or Continue
Insert/Update General Characteristic Data
Characteristic Identification and Association Resources
Present State of Build Resources
Build Resources
Support Resources

Figure 11. Formulator Internal View

Provide capability for update of all evaluations made prior to latest release of an updated description_area;
-- Request KB to check_and_report_and_update-if_possible on any evaluations
-- using old description_area.

Provide capability to insert semantic_net_description_data and frame data (or characteristic data);
-- Calls to the KB through Characteristic Identification and Association resources
-- and Build resources to request methods: get_and_put_frame_data, and
-- get_and_put_semantic_net_builders
-- (like create_link, eliminate_link, reuse_sub-semantic_net_structure,
-- eliminate_sub_semantic_net_structure).

Provide capability to maintain current_state of Formulator process;
-- Calls to KB to "update_present_state_summary" through Present State of Build
-- resources.

Provide capability to traverse_forward, backward, or at the same_level in a semantic_net for review/rework;
-- Calls to KB through Build resources to request: work_at_present_level,
-- work_at_next_level_down/up, show_all_not_processed/completed_frames
-- go_to_not_processed/completed_frames.

Provide capability to get a printout of work accomplished;
-- Calls Support resources for method "print report".

Provide ability to accept-partial/accept-complete/abort the session;
-- Calls KB through Present State resources to: Accept_session, Abort_session,
-- Check_if_semantic_net_is_completely_defined

Design Discussion of New Evaluator . Both the Evaluator and the Selector sections follow the format of the Formulator section. The new Evaluator internal view is shown in Figure 12. Its processes and interfaces or resources are enumerated in the figure. The Evaluator Process conceptually exhibits the behavior as described in the requirements document (with the improved behavior identified in the analysis section superseding weaker behaviors) and it achieves this behavior using an object oriented methodology. To understand how this processor accomplishes its job, the following Top-level Structured English describes the processing that the new Evaluator must accomplish.

Initialize to Evaluator_start_state;
-- Call KB routine "Initialize_KB_evaluator" through Present State of Evaluation
-- initialization method. Set Evaluator level to novice

Request Evaluator and Tool information;
-- Use Evaluation resources to call "Get_evaluator_data" and "Get_tool_data"
-- methods from KB.

Identify correct SAI;
-- Calls to the KB through Characteristic Identification and Association resources
-- requesting to provide method "identify_SAI".

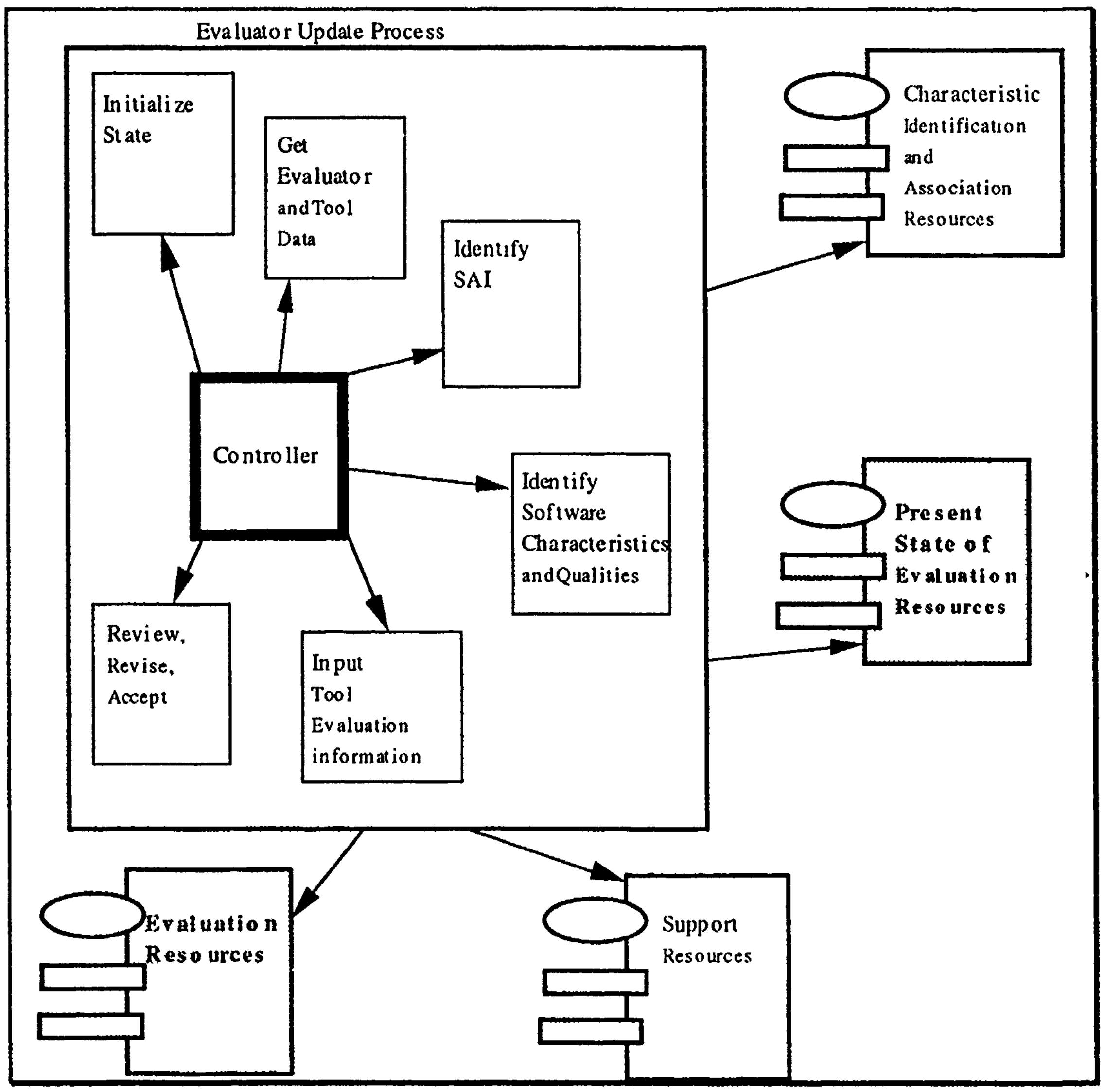
Evaluator Update Process
Initialize State
Get Evaluator and Tool Data
Identify SAI
Controller
Identify Software Characteristics and Qualities
Review, Revise, Accept
Input Tool Evaluation information
Characteristic Identification and Association Resources
Present State of Evaluation Resources
Evaluation Resources
Support Resources

Figure 12. Evaluator Internal View

Provide for ability to toggle Evaluator_processing_level;
-- Novice level, verses Expert level, Request method "Define_how_to_proceed"
-- from KB through Evaluation Resources.

Provide characteristics for evaluation;
-- By requesting that KB
-- "Provide_Characteristics_at_Evaluator_processing_level" through
-- Characteristic Identification and Association resources

```
Direct user on how to evaluate;
   -- Request KB provide method "how_to_evaluate_characteristic" through
   --       Evaluation resources.

Store evaluation data;
   -- Call method "Store_characteristic_info" from KB through Present State of
   --       Evaluation process.

Provide ability to review/change session inputs on request;
   -- Call KB to "provide_evaluation_process_summary" through Present State
   --       Resources.

Provide capability to Accept, Abort or report_on session;
   -- Call method "Commit_Database" or "Rollback_database" from KB through
   --       Evaluation resources or call Support Resources for "provide_report".
```

Design Discussion of New Selector. The new Selector internal view is shown in Figure 13. Its processes and interfaces or resources are enumerated in the figure. The Selector Process conceptually exhibits the behavior as described in the requirements document (with the improved behavior identified in the analysis section superseding weaker behaviors) and it achieves this behavior using an object oriented methodology. To understand how this processor accomplishes its job, the following Top-level Structured English describes the processing that the new Selector must accomplish.

```
Initialize to Selector_start_state;
   -- Call KB routine "Initialize_KB_selector" through Present State of Choices
   --       initialization method. Set Selector level to novice

Identify correct SAI;
   -- Calls to the KB through Characteristic Identification and Association resources
   --       requesting to provide method "identify_SAI"

Provide for ability to toggle Selector_processing_level;
   -- Novice level verses Expert level, Request method "Define_how_to_proceed"
   --       from KB through Present State of Choices Resources.

Provide characteristics for selection;
   -- By requesting that KB
   --       "Provide_Characteristics_at_Selector_processing_level" through
   --       Characteristic Identification and Association resources

Direct user on how to go through selection process;
   -- Request KB provide method "how_to_proceed" through Support resources.
```

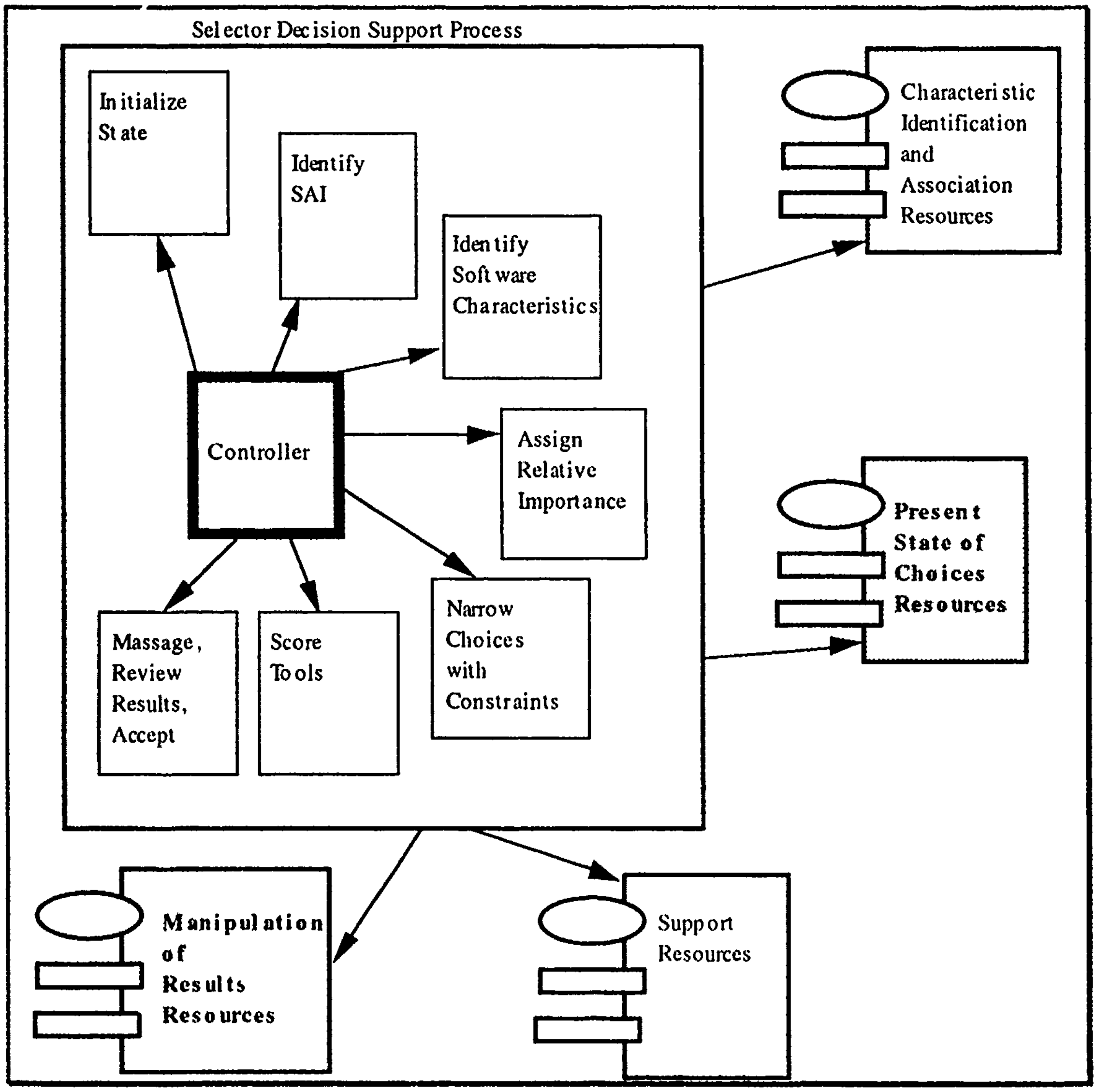
Selector Decision Support Process
Initialize State
Identify SAI
Identify Software Characteristics
Controller
Assign Relative Importance
Massage, Review Results, Accept
Score Tools
Narrow Choices with Constraints
Characteristic Identification and Association Resources
Present State of Choices Resources
Manipulation of Results Resources
Support Resources

Figure 13. Selector Internal View

Store selection weights data;
-- Call method "Store_characteristic_weight_info" from KB through Present
-- State of Selection process.

Provide ability to score tools ;
-- Call KB to "Score_tools" through Present State Resources.

Provide ability to review session inputs on request;
-- Call KB to "provide_selection_process_summary" through Present State
-- Resources.

```
Provide ability to re-accomplish selected weights and criteria to achieve different
   results;
   -- Call KB to "reset_cutoff_threshold", re-accomplish criteria or
   --       re-define_essential_characteristic processing through Manipulation of
   --       Results Resources.

Provide capability to Accept, Abort or report_on session;
   -- Call method "Commit_Database" or "Rollback_database" from KB through
   --       Manipulation of Results resources or call Support Resources for
   --       "provide_report".
```

Functionality of Rest of OOD Design. With the system processors defined all that is left to describe is the functionality of all remaining interfaces (see Figure 10). The usual functions are provided by the User Interface resources and the SQL Interface resources. The User Interface provides all resources necessary to provide and acquire information to/from a STEMdB user. The SQL interface provides all resources necessary to store, update and retrieve information from a commercial database. The Knowledge Base interface provides all methods and types that a processor must have to operate with the system data's structure and content. It provides the functionality of creating structures in the database to store system data and it provides resources to store and retrieve system data from the database. It provides the resources to maintain a running summary of what characteristics/structures were accessed and how they were modified. It provides the raw scores (quality and functionality) of tools to the Selector (the Selector maintaining its own cutoff threshold and top level weights accomplishes further processing on this data). It provides the build resources to the Formulator and it provides the insert and delete characteristic data/weights resources to the Selector and Evaluator. It provides all characteristic viewing resources and all structure traversal resources to all processors. Specific viewing resources are implemented within each processor under the Support Resources Package.

Top Level Design of an Abstract System Interface to an SQL Database

Chapter III established that STEMdB portability and maintainability were hindered by dependence on one commercial database and it identified the need to design the entire STEMdB in Ada. To make the STEMdB independent of any database an abstract interface needed to be created and incorporated. Hidden complications always seem to arise when one tries to create an interface from one application to another, however. For instance, in creating an interface between the two procedural languages Ada and C, a binding designer must understand the design foundations of both languages and he/she must create conversion routines to avoid conflicts that arise from design differences. One example of a design difference between Ada and C is: C has null terminated character strings and Ada does not. Creating a binding or interface between a procedural language like Ada and a non-procedural or data oriented language like SQL complicates interface designs even more. Extensive work has already been accomplished towards identifying these complications, and a model complete with template resources was developed to address this exact type of interface development. This model and the methods associated with it are called the SAME [17]. The SAME is a binding or interface between Ada and SQL that allows both Ada and SQL to accomplish their jobs without compromising each other's design foundations, while at the same time allowing Ada's abstraction mechanisms to overcome some of SQL's shortcomings. It allows for the safe treatment of SQL null values and it makes extensive use of the Ada exception mechanism.

Problems Addressed by the SAME. Before proceeding on with an overview of the SAME method, it may help to understand the interfacing problems it was developed to overcome. Graham in [17:17] identified five problems specific to Ada to SQL bindings that his model the SAME addresses. Those five areas were:

1. Typing model differences between Ada and SQL - The major difference between typing models is that Ada has an abstract typing capability while SQL does not. In fact, SQL is in the opposite end of the spectrum when compared with a robust typing model since it operates on a limited set of types.

2. Treatment of null values - Ada works only in a two valued logic world where SQL uses three valued logic. For instance, SQL provides logical operations that expect variables to be in the form of "true, false, or null". Ada can only logically operate on variables that are either true or false.

3. String Processing - SQL pads strings with blanks, and its character sets are database implementation defined. Ada uses the predefined character set called ASCII for package standard operations.

4. Decimal Fixed Point Arithmetic - The operations on decimals in both Ada and SQL work differently. SQL implementations store decimals in a packed machine format, Binary Code Decimal (BCD), which Ada does not recognize.

5. Types defined outside of the SQL standard - Commonly used types such as the Date type are important to model yet they are implemented in different ways. Graham also identifies the need to be able to store enumerated types in SQL when SQL does not support an SQL enumeration type.

This research will use the SAME model to overcome all problems identified in this listing. It will be presented as a top-level design, however, and solutions to some of these problems may not be obvious until a more detailed level design is accomplished. With this justification complete, an overview of the SAME method and a STEMdB interface design using the SAME can be presented.

Overview of How to Apply the SAME Method. Graham builds the SAME method from the bottom up. He builds his abstract interface based on primitives he calls abstract domains. An abstract domain in the SAME model is identified by a unique name given to a column name in a table. The common sense rule that designers must incorporate when creating these abstract domains is: if two distinct columns represent the same type of information and they can be compared to each other, then their abstract domains are the same. Graham uses the example of one application having two distinct tables in which each has a "city" attribute or column name. Since they both represent the same abstract domain only one domain primitive is defined, and it is named "CITY". [17:8-10]

Once these column name equivalents are identified they are associated with two types, a null bearing and not null bearing, and all the methods that operate on those types. This association is necessary to model SQL null values in Ada and to define specific types that represent SQL attribute objects. In general, this association occurs as a two step process. First the null bearing type for a given column_name is assigned the name, "column_name_**Type**" and the not null bearing type for the same column_name is assigned the name "column_name_**Not_Null**". Then these names are used to create an Ada derived type that is based on the SAME standard package that maps to the correct type of SQL domain. An instantiation of a generic operations package along with these derived types produces a Domain Primitive Abstract Type. Figure 14 shows these building block abstract domains as "Domain Primitive Abstract Types". This figure is presented to clarify the foundation upon which the SAME typing model is built. The way Graham builds this typing model is through the use of "Ada Derived Types". Using a derived type in Ada creates a new base type. In Figure 14 the Concrete Types and the Domain Primitive Abstract Types are derived from their foundational types. A foundational type in Figure 14 is the type immediately below a given type (for instance, SQL-Based Pre-Defined Types are the foundational type of Concrete Types). The package called SQL_Standard defines "SQL Based Pre-Defined Types" as: Char, Smallint, Int, Real, Double_Precision, Decimal, SQL_code_Type, Sql_Error, Not_Found and Indicator_Type. It defines the majority of these types by placing database implementation tailored constraints on Ada predefined types. As an example, Type SQL_Standard.Int is defined in the following way: **type Int is range bi..ti;**. The place holders "bi" and "ti" are the actual integer upper and lower limits defined by the database implementation. These actual limits have to be inserted in place of "bi" and "ti" when installing the SAME resources. The Concrete Types are then built on top of these SQL Based Types by defining derived types and operations for each of the basic pre-defined type equivalents. The following is a listing of packages that define these Concrete Types: SQL_Char_Pkg, SQL_SmallInt_Pkg, SQL_Int_Pkg, SQL_Real_Pkg, SQL_Double_Precision_Pkg, SQL_Decimal_Pkg. Appendix B has a complete copy of the SAME supplied package specification resources discussed in this research [17:143-248]. It is not the intention of this research to describe how to set up the SAME environment but rather how to apply it. The reader is directed to [17] for details.

With all of this typing information understood (with the exception of Composite Domain Types which will be explained within the next few paragraphs), it is now important to present an example relative to the STEMdB of how to define a domain primitive type. To set up the abstract domain primitive type for TOOL_ID the following package would have to be created:

```
with SQL_Int_Pkg;
package Tool_id_primitive_domain is
  type TOOL_ID_Not_Null is new SQL_Int_Pkg.SQL_Int_Not_Null;
  type TOOL_ID_Type is new SQL_Int_Pkg.SQL_Int;
  package TOOL_ID_Ops is new
     SQL_Int_Pkg.SQL_Int_Ops(TOOL_ID_Type, TOOL_ID_Not_Null);
end Tool_id_primitive_domain;
```

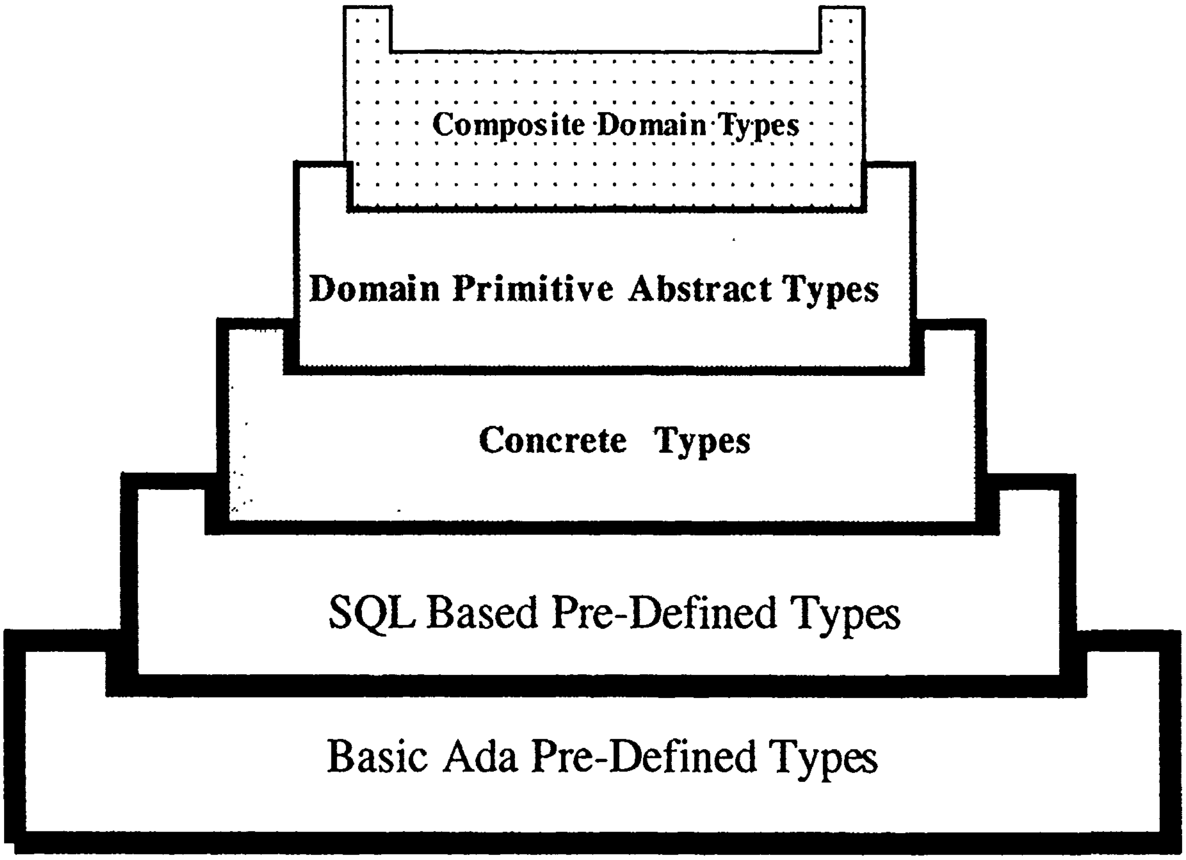
Composite Domain Types
Domain Primitive Abstract Types
Concrete Types
SQL Based Pre-Defined Types
Basic Ada Pre-Defined Types

Figure 14. SAME Foundational Types

All primitive abstract domain dependent operations are defined by the instantiation of the "SQL_*_Pkg.SQL_*_OPS(...)" generic packages (where "*" represents a wildcard placeholder and in the above example it would take on the value "Int"), all other operations are inherited from the derived type's domain. These primitive dependent type operations define how to get a null type given a not_null type and vice versa . They also define "Assign" operations for the limited private types which define all null bearing types. For instance, TOOL_ID_Type is a limited private type which could not be assigned if an assign operation was not defined for it. Use of the limited private types to define a null bearing type is necessary since a null bearing type is a two component record that contains a not null bearing type and a Boolean that represents whether the record's contents are null or not. At any given time if the record's state (as determined by the null component) is not null then the contents of its value part component are valid, otherwise the contents are invalid.

The goal of all of these building block types and operations is to support the creation of the "Composite Domain Types" shown in Figure 14. These composite domain types are simply combinations of domain primitive types that together represent some object in an abstract interface. These composite types are usually defined by a record structure containing the primitive types. The operations associated with that record structure, once defined, complete the definition of an abstract interface to an "SQL implementation module".

It is important at this point to diverge and explain how an SQL implementation module can be built. Since there were no known standard SQL module compilers available at the time of this research, a substitute method for producing SQL module resources had to be found. The substitute method chosen was to create database supported SQL modules by implementing an SQL module using the embedded SQL calls within a supported language framework. This would require creating another interface between Ada and the database supported interface language in order to call modules in that language. For example, the Oracle Database for the Macintosh supports embedded SQL calls from within the framework of a C program [25]. Meridian Ada for the Macintosh supports pragma interface calls to Macintosh Programers Workshop C (**MPW C**) object libraries. To implement an abstract interface between Meridian Ada and Oracle on the Macintosh, MPW C resources would have to be built using database supplied embedded SQL resources. These resources would then have to be pre-compiled and compiled before the body of the Ada abstract interface would be able to call them using a newly designed MPW C interface and an Ada Pragma Interface.

Since the goal of this research is to define an abstract interface for an Ada to SQL binding, the details of the bodies of those abstract interfaces (which are implementation issues not top level design issues) will not be described other than to describe an example of their general structure and desired behavior.

It is now time to take another look at the definition of a SAME abstract interface or binding, and the operations that make up that definition. Graham emphasizes that application logic should never enter into the definition of an abstract interface's operations. He then states that application logic should be built at least one layer above the abstract interface. This is sound design advice since the design of the interface has the single goal of communicating with a database. Graham also emphasizes that the database should be allowed to accomplish the operations that it is best suited for while the body of the abstract interface's main goal should be to work as a translator. In general, the operations that are defined in the abstract interface mimic those that must be called to manipulate a structure or table in a database. These operations can be single record based and called by defining SQL procedure call interfaces or they can be "cursor based". [17:59]

Cursor based calls are the mechanism that allows an application to work with multiple record retrievals. A cursor is defined in SQL as a group of records (defined by the cursor's SQL statement) that can be opened, stepped through with a fetch operation, and closed. A Cursor is defined by its cursor declaration which contains exactly one SQL statement. The Cursor becomes visible to the application only after it is opened, and it can only be operated on within a running application while it is opened. A cursor can only be stepped through in one direction, the only way to return to a previously fetched record is to close and re-open the cursor. A database can have multiple cursors opened at the same time and these cursors can be identified by their cursor names.

Graham identifies all possible basic database operations or SQL statements that may need to be modeled in the abstract interface in [17:60]. Table 3 provides this list along with an identifier column which classifies whether the type of statement is a transaction (T), a cursor (C) or a non-cursor (NC). According to Date in [15:48-49] transactions, cursors and non-cursors are the three classifications that all SQL manipulative statements fall into. The "Ada Parameter Kinds" column lists two types of parameters which the Abstract interface will use when making calls to an SQL module. Graham defines the "row record" as an object of the Composite domain type and the "individual parameters" as objects of Domain primitive types. He explains that the individual parameters will be used mostly when an interface designer must model SQL "having" or "where" clause information. Both of these clauses are used in SQL to qualify the type of data desired. The "where" clause is used to eliminate rows in non-grouped select statements, the "having" clause is used to eliminate rows in the "group by" select statements [15:95].

The last area that needs to be addressed before presenting a design of the abstract interface of the STEMdB using the SAME framework is how the body of the interface is expected to behave. There are four behavioral requirements that this implementation must meet. It must convert any inputs to an SQL module from Abstract domain primitive types to SQL_Standard types, it must call the SQL module, it must convert all necessary outputs of an SQL module back to Abstract domain primitive types and it must perform error checking using both SQL indicator parameters and the Sqlcode parameter. Indicator parameters in SQL are used to indicate if a fetched field is null (the indicator is a Boolean that is set to true when nothing matches the database call's criteria). Sqlcode parameters are used to indicate implementer defined database errors associated with database operations. [17:62]

Table 3. SQL Statement to Ada Mappings

Type	SQL Statement	Ada Parameter Kinds	Mode
C	close	none	
T	commit	none	
C	positioned delete	none	
NC	searched delete	Individual Parameters	in
C	fetch	row record	in, OUT
NC	insert (values)	row record	in
NC	insert (subquery)	Individual Parameters	in
C	open	Individual Parameters	in
T	rollback	none	
NC	select	row record & Individual Parameters	in, OUT in
C	positioned update	Individual Parameters	in
NC	searched update	Individual Parameters	in

LEGEND
C = Cursor Operation
NC = Non-cursor Operation
T = Transaction Operation

STEMdB Abstract Interface Design Using SAME. Accomplishing the design of the abstract interface was a three step process. The first step involved defining the architecture of the domain primitive types, the second involved identifying basic database operations that each of the three STEMdB processes would need to accomplish its job, and the third involved identifying the composite domain types, individual parameters and associated operations that would support these higher level STEMdB operations.

During this process it became obvious that the STEMdB could not be feasibly designed without utilizing the additional SQL methods and types associated with dynamic SQL2. SQL2 is an extension standard that was being developed in 1989 to address areas where the SQL standard was weak or lacking [17]. The STEMdB design requirement that the user be allowed to randomly select anywhere from one to eleven desired attributes to narrow down tool selection candidates [19:17] was the deciding factor for studying a dynamic approach. This requirement would force a static SQL design to provide one select operation for every possible combination of all eleven inputs. This would mean that 2048 operations would have to be designed and supported. To avoid this unnecessary coding, the dynamic SQL2 design addressed by Graham was used. [17:117-125]

Graham presented two methods for approaching a dynamic design. Both methods were based on the "<dynamic using description area structure> or SQLDA" [17:115]. This structure supports a buffered approach that many databases use to dynamically communicate with application programs. Graham asserted that the two methods differed in their visibility of the SQLDA structure from an application program's perspective. The first method duplicated the SQLDA structure on the abstract interface side, and the second method applied a functional approach which hid the details of the SQLDA on the database module side. The overhead associated with the "visible" method was considered to be too much by Graham, so he advocated the functional method. His arguments about duplication of data and multiple transformations slowing down an interactive session were well founded and the functional approach was used in parts of the STEMdB abstract interface design.

The functional approach to dynamic SQL2 required two additional resource packages, SQL_Standard_Dynamic and SQL_Dynamic_Pkg, copies of which reside in Appendix B. The assumptions made to simplify this approach were the same ones used in Graham's example in [17:124]: only one dynamic statement would be in use at any given time and its contents would be available in an object called STMT of SQL_Char_Not_Null type. There were two top level functionalities that needed to be provided by a dynamic interface. First the interface had to allow for the set up of desired statement instantiations and then it had to provide for operating on those instantiations using parameter type knowledge.

Before proceeding with the overall design discussion, the actual process of using a dynamic interface will be presented. This should help the reader to understand why certain functions and procedures must be provided by the dynamic interface. The process that had to be followed from an application standpoint using [17:124-125] as a guideline was: prepare the STMT, allocate a name for both input and output SQLDA processing while associating a maximum number of parameters with each name, create the link between these names and the prepared STMT by calling a Describe function, provide a get_parameter_count function that operates on a SQLDA name, and use this function to check if there are any input parameters (this is all based on the Describe function using the prepared statement as a template for building an SQLDA instantiation complete with parameter types that are waiting to be filled with objects). If there are inputs, step through each input parameter and, using a get_parameter_type function, obtain the parameter's type. Then use the type in a "case" statement to pick the correct set_parameter_value function to insert the dynamic SQLdata object into that parameter (the dynamic SQLdata object is obtained from the user as a criterion that is desired to hold true for that parameter). Once all inputs are processed begin output processing. Use the same get_parameter_count function to discover how many output parameters there are in the Output_SQLDA object. If there are zero outputs then it is not a select operation, so execute it. If there are outputs, open a cursor that is associated with the Input_SQLDA and process the cursor with associated Fetch , Get_parameter_type and Get_SQLdata functions (which are selected based on the SQLType retrieveu, then close the cursor. All of this processing can be accomplished if the dynamic abstract interface provides the eleven procedures/functions identified in Table 4.

Table 4. Functions Provided by an Abstract Dynamic Interface

Prepare()
Allocate()
Describe_Input()
Describe_output()
Get_Number_parameters()
Get_parameter_type()
Set_parameter_value()
Get_paramter_value()
Open_Cursor()
Fetch()
Close()

With the discussion of dynamic interface considerations complete, the overall design approach can now be presented. The design will be described in the context of three layers, the Primitive layer, the Database intelligent layer, and the Application intelligent layer. A description of all of these layers will be provided as the design discussion progresses.

Instead of having each of the lowest level domain primitive abstract types as a package, this researcher decided to group types into entity level packages. This helped to reduce the number of domain primitive packages that would need to be "withed" or made visible for both further abstract interface design and application program design. It also encapsulated entity and relationship information. All non-trivial relationships attributes were also grouped into their own domain abstract primitive type packages. Whenever relationships had "foreign keys" ,which are keys inherited from connected entities, the keys would already be defined in the connected entity's domain primitive definition (i.e., they were not re-defined in the relationship domain). Any operation that worked on a relationship domain would have to "with" all appropriate connected entity domains along with the relationship primitive domains. For instance, a formulator operation of linking a domain with a GSC would require an update to the root_node table which would require an operation on the domain primitive types defined in the root_node. There are no primitive types defined for the root_node relationship because all attributes are foreign keys and they are already defined in the connected entities domain primitive type definitions. The update operation would have to have visibility to both The_Area and the GSC domain primitive types. The domain primitive type packages are defined in Appendix C. To clarify how each attribute contributes to this package, each attribute's information is consolidated in one area which is separated from other attribute information areas by a blank line. To implement these packages all generic "SQL_*_Pkg.SQL_*_OPS(...)" packages must be moved to the end of each entity package since they are later declarative items (later declarative items are defined in the Ada Language Reference Manual).

One more design grouping of abstract domain primitive types was considered. By looking at the highlighted entities and objects in Figures 15, 16 and 17 (Note: When possible, these figures reflect the design changes advocated in Chapter III: for instance the the linkST relation is replaced by the linkSAWT relation.) one realizes that application programs responsible for altering the state of the highlighted entities and relationships will be written for each of the three subsystems. Therefore, the Ada package could be used to group the domain primitive abstract type packages that were created from the highlighted entities and relationships into three disjoint packages. It was decided that this design encapsulation would be better utilized in a layer of abstract interface that is built on top of the primitive interface to the database (which is being derived now). For clarification purposes this new blanket layer will be called the "Database intelligent layer". The Database intelligent layer design is discussed in concept at the end of this section but is not addressed any further by the design since it is more detail than the top level design of this chapter required. A top level interface design in the sense of this chapter answers the question: "what is the minimum required to get the interface to the database defined without bringing in application logic?".

To streamline access to domain primitive types, the idea of placing composite methods built on domain primitive types into the same entity packages that encapsulated those types was considered. Trying to place composite methods into packages that also defined their foundation types meant that composite types (or records) and procedures would have to be declared based on primitive types that were not fully defined. This would be true because the instantiation of generic packages "SQL_*_Pkg.SQL_*_OPS(...)" is a "later_declarative_item" according to the Ada language reference manual and it must be defined after all basic declaration items. Since attempting this would violate the rules of Ada and would not work, this idea was aborted.

A better solution places the composite operations into entity packages that mirrored their foundational domain primitive type entity packages. The basic behavior that all of these packages would provide would be in the form of "inserting, updating, deleting, searching and retrieving" objects of attribute types and composite record types. In addition to these composite operation packages which would be entity specific, two additional packages would be required, one to model dynamic interfaces and one to work on database transactions. Neither of these two additional packages would provide operations based on a single entity or relationship, yet operations they contained would be necessary to complete a definition of the primitive layer. This justifyed encapsulating these operations in their own packages. Specifications for all of these packages are located in Appendix D.

Before continuing on with a description of the design, it is necessary to first define the difference between logical packages and physical packages in the context of this document. A physical package is what would actually be coded as an Ada package by an implementer. A logical package allows information to be grouped so that the concepts explained in this design are more easily understood. Physically, the logically grouped packages would still remain separate and distinct packages. One other concept that needs explaining is the idea of having "view packages". Essentially a view package provides all top level knowledge base operations and types to its respective subsystem (All view packages taken together represent the Application intelligent layer.) . It operates similar to the way views operate in database applications. It gives the calling subsystem the minimum information needed to get the job done while hiding the details of the operations and types.

With the basic building blocks established and an explanation of views and logically grouped packages complete, the perspective of how this all fits in with an object oriented STEMdB design can now be presented. First, each domain primitive type package and composite operation package that operates on the same entity or relation is logically grouped into one package that is defined by the entity's name. All logically grouped entities conjoined with the transactions and dynamic packages represent the Primitive layer. This Primitive layer is the foundation upon which the Database intelligent layer is built.

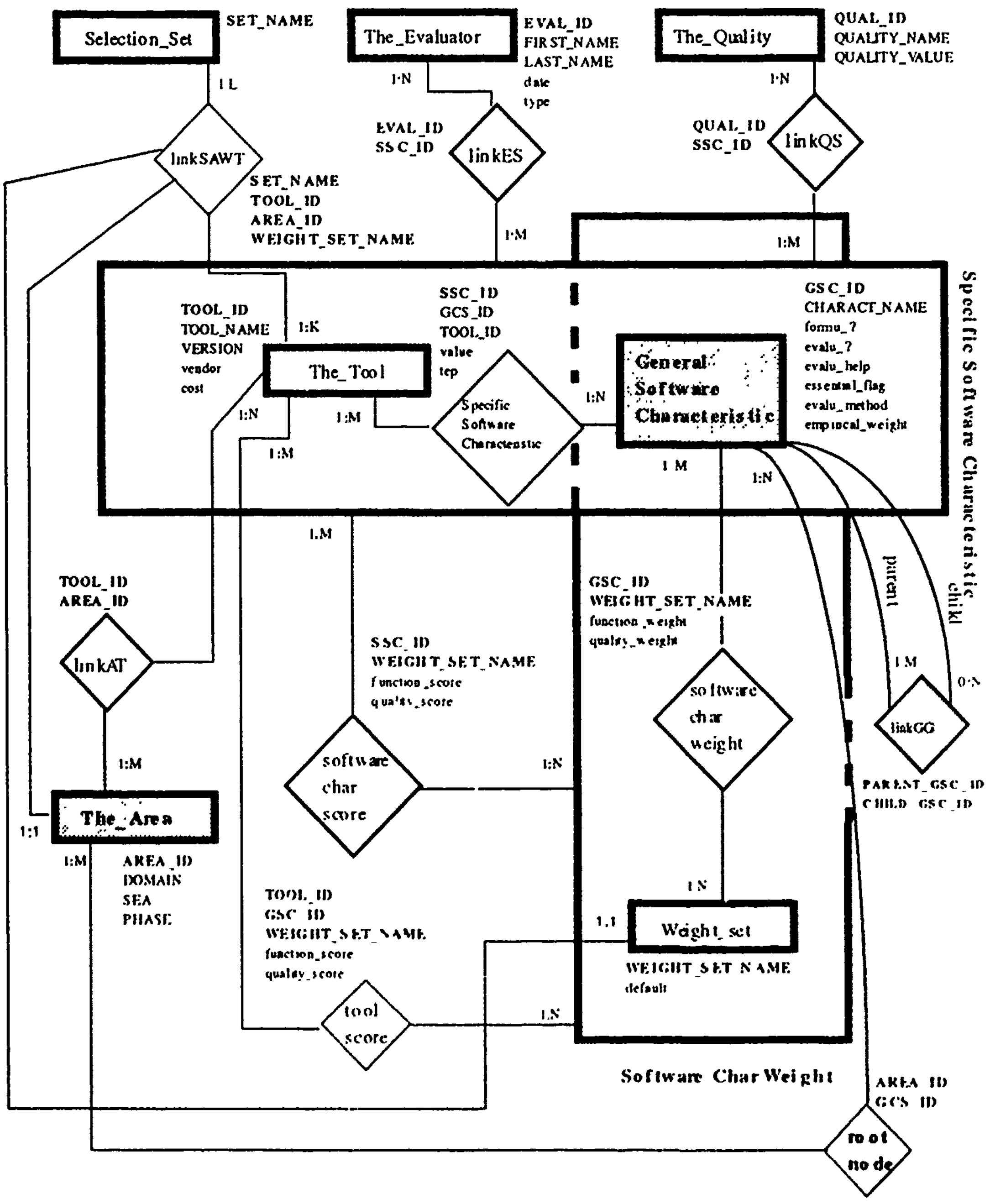

Figure 15. Formulator Altered Objects

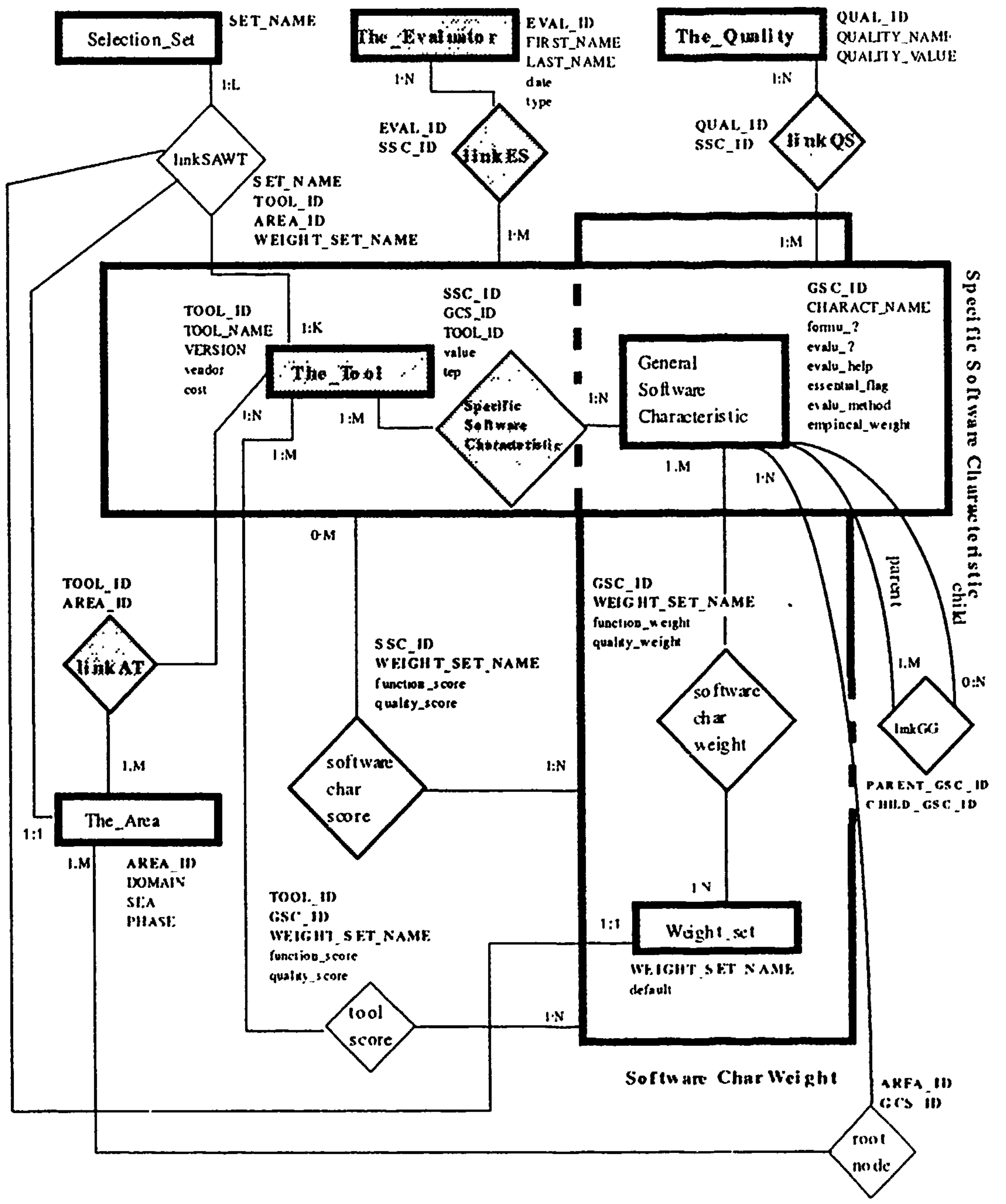

Figure 16. Evaluator Altered Objects

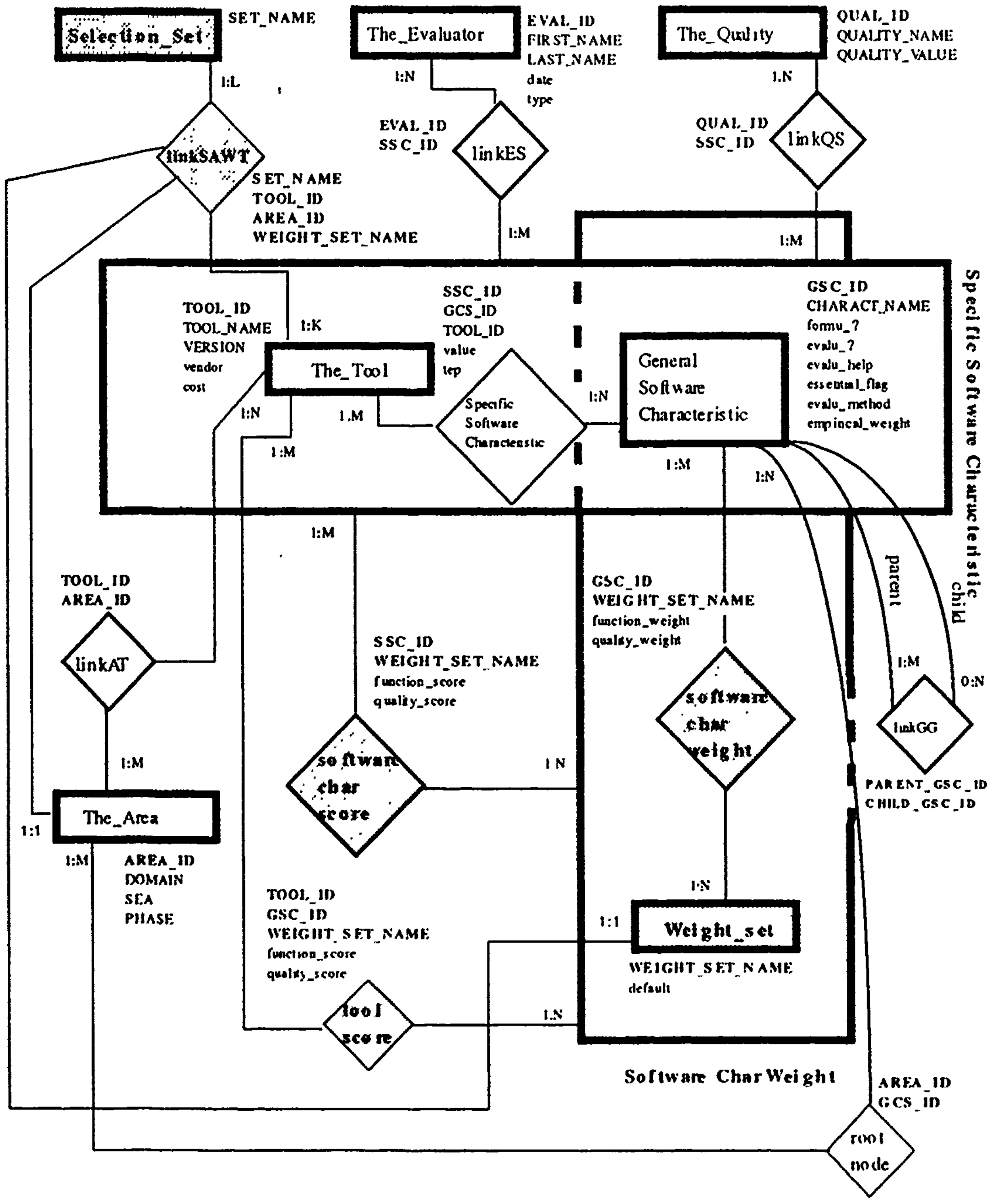

Figure 17. Selector Altered Objects

The Database intelligent layer can be separated into three distinct physical packages (justification for this approach was introduced earlier in this section) which each encapsulate operations that work on entities belonging to a group. The entity groupings are associated with the state altering behavior of the calling subsystems. Any entity that can be physically altered by a calling subsystem has Database intelligent layer operations defined in a package that is named after that subsystem. For instance, the Formulator can alter the states of the General Software Characteristic, the linkGG, the root_node and The_Area entities and relationships (see Figure 15), so the operations for these objects are located in the Formulator Altered Grouping package which is contained within the Database intelligent layer. Next level non-state altering operations for these entities are co-located in the same package. Figure 18 shows how this composite build process would be accomplished for Formulator State Alteration considerations (the Evaluator and Selector considerations would be built using the same method). Essentially the single overall desired behavior of the Database intelligent layer is to provide the database application operations that use the resources provided within the Primitive layer. For instance, to acquire all GSC_ID's for a given SAI, the Database intelligent layer would know only how to open a cursor, step through a loop which calls a fetch_gcs_ID operation in the Primitive layer and close that cursor when done. It would know when the fetch operation is complete by using a Boolean check that is updated on each fetch by the Primitive layer.

The more sophisticated or complex interface logic occurs at the "view level" within the Application intelligent layer. As noted earlier, the Application intelligent layer is composed of three view packages: Formulator View, the Evaluator View, and the Selector View. The Database intelligent layer interfaces would be visible to these subsystem view interfaces which in turn would be visible to the subsystems (all through the specifications or well defined interfaces, of course). The Database intelligent layer methods in Figure 18, for example, would be callable by the view interfaces tailored towards each of the three exterior subsystems. Figure 19 illustrates how the view logic in the Knowledge Base would call the abstract interface to accommodate requests made by the other subsystems.

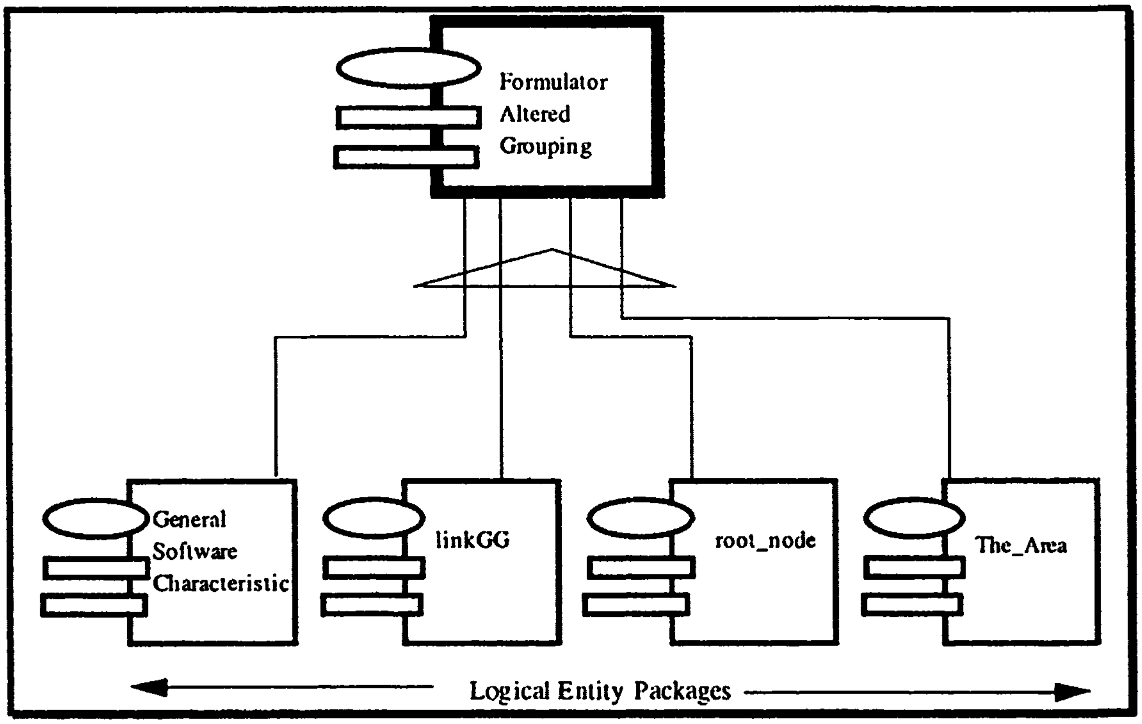

Figure 18. Type Build Process. Formulator State Alterations

With the big picture now complete, a word of caution is in order. The reader is cautioned at this point that before any attempt is made to design applications for any level abstract interface the issues of type conversions and visibility of an "Abstract domain SQL base type" must be understood. Since writing the applications that use these SAME abstract interface types is outside of the scope of this research, application program type conversions will not be addressed further. The reader is referred to [17:69-76] for a detailed discussion of these issues.

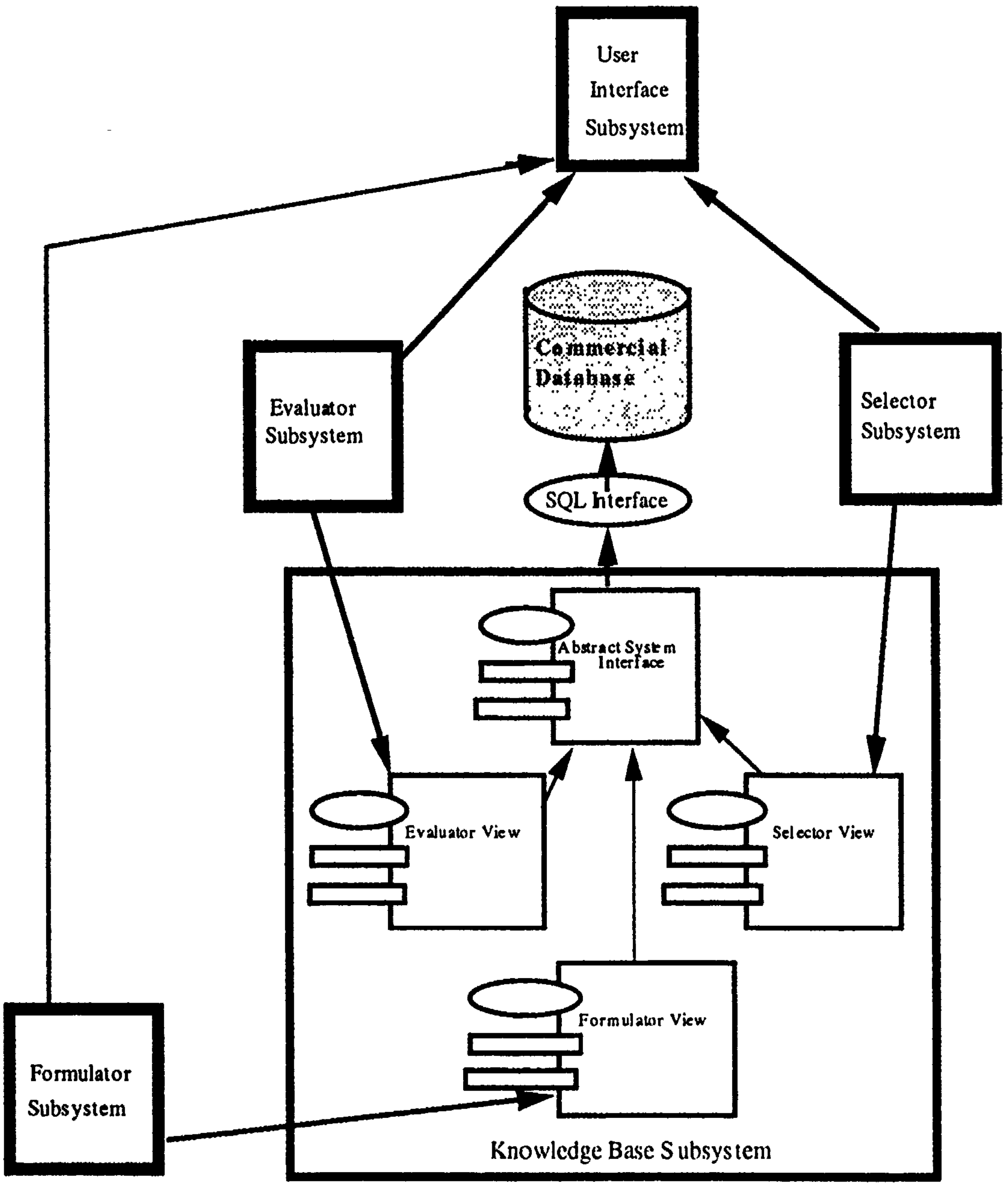

Figure 19. Application Logic Calls to Abstract Interface

Summary

The goal of this chapter was to emphasize how the STEMdB could be re-designed to be more robust, maintainable, upgradeable and portable. It provided a top level overview of how an object oriented approach to the STEMdB design could be accomplished. Using an object oriented approach to this design helped to accomplish the goals of a more robust and maintainable design. The chapter also provided a method for creating an abstract interface with a database and it presented a design of a STEMdB abstract interface using that method. It established that database independence can be achieved through abstract interface design. By achieving database independence the design is made more portable. The only other issues that could hamper the design's portability are its dependence on a commercial user interface and its dependence on a commercial database's dynamic SQL capabilities. The designers of the STEMdB should try to isolate the user interface and dynamic SQL operations so that the impacts of these dependencies are minimal and localized.

V. Conclusions and Recommendations

Overview

This chapter provides a summary of the accomplishments in this research. It also provides recommendations for re-accomplishing the STEMdB design and for further work in the area of developing abstract interfaces to databases using Ada.

Summary of Research

This research began by surveying the literature for information on what a CASE tool was and how to evaluate a CASE tool for a given domain. The E&V Guidebook, IEEE working group studies and the Lawlis dissertation provided a solid background in these areas. The Lawlis dissertation also provided an automated framework for supporting decision support CASE tool selections. Using these resources, the goal of this research was to analyze the prototype development effort, the STEMdB, and to provide constructive ways that the prototype could be improved.

Initially, the only prototype tool information available was a working prototype (along with its source code) without design documentation. This prototype was developed on a Macintosh environment called MacApp, and it was implemented in Think Pascal. Non-availability of this environment and unfamiliarity with Think Pascal made a reverse engineering effort difficult . Documentation on resources "withed" from the MacApp environment had to be acquired in order to understand the design. A requirements and design document for the next phase prototype solved documentation problems and a thorough analysis could be made given what this document did and didn't say and given the behavior of the initial prototype. To provide a basis for comparison, the ASSIST prototype had to be used and understood.

The ASSIST prototype was developed on the Hypercard environment using the Hypertalk language on the Macintosh. Documentation in the Lawlis dissertation provided a lot of information about the tools behavior, but it did not provide enough information on design. It provided object diagrams and a thorough description of resources provided as well as behavior exhibited, but knowledge about how all the objects interacted in the form of a driver was lacking. To gain a better understanding of the design Hypercard and Hypertalk were studied. The intricate and nested nature of Hypercard scripts and cards made it very cumbersome to accomplish a top level reverse engineering effort on the ASSIST prototype. The goal of this effort was to see how the tool accomplished its behavior. This knowledge would help in understanding the concept of a Knowledge Base which was the foundation of the ASSIST design and which later proved to be a good foundation for the STEMdB design. Once all of this information was understood it was used to measure what the STEMdB's limitations were given that it was not using a Knowledge Base approach.

Once an analysis of the STEMdB was accomplished, it was discovered that a significant improvement in tool portability could be achieved by accomplishing an abstract interface between the tool and its database. A third background analysis had to be accomplished in the form of application programs working with commercial databases through an interface in order understand some of the details in this type of approach. Oracle for the Macintosh was an implementation that was available, so it was studied to acquire this background. Information in the form of the SAME documentation provided domain knowledge on Ada to SQL bindings that made it possible to specify a STEMdB abstract interface.

Conclusions

This research covered several different domains to accomplish the goal of analyzing and suggesting improvements to the STEMdB prototype effort. The necessity for a DSS tool for CASE tool selection will continue as long as CASE tools are used to aid in software development efforts. CASE tool automation is essential to maintain control and consistency over large software projects. The STEMdB provides a means to sift through the information describing these CASE tools and to help the decision maker make an informed decision. The STEMdB's biggest drawback is that it is targeted towards being operated only by its developing organization. Concentrating on only the developing organization as an end user allowed the designers to make decisions that restricted the design's capabilities. The tool's wide-spread need throughout the DOD should be taken into account and the tool should be developed to support remote users. By incorporating the design changes proposed in this research, the STEMdB can be made available to a larger group of users, and can be made more robust, maintainable, and portable.

The design can be made more robust by encapsulating knowledge within the Knowledge Base. Any design decision to alter the implementation of anything that is provided as a method to the exterior three subsystems will affect only the body of the package that defines that method These bodies are encapsulated within the Knowledge Base subsystem, so design changes like this would only require a re-compilation of the Knowledge Base subsystem affected packages (assuming Ada becomes the language of choice). The ability to change these implementations while not affecting the rest of the system makes this design robust.

Given that the need for such a DSS will persist as long as CASE tools remain popular, a DSS for CASE tool selection has the potential to be around for decades. Whenever software tools have the potential to exist that long, they should be designed to be as maintainable as possible. The existing design allows for tight coupling between the three subsystems. By designing the STEMdB as an object oriented system which isolates the objects most likely to change in a Knowledge Base subsystem this coupling can be eliminated. Providing well-defined interfaces in the form of methods and types, de-couples the design, which supports maintainability.

By designing an abstract interface to the database, the portability restrictions (resulting from a tight coupling) placed on a design targeted for one specific SQL database are lifted. This interface also allows the Ada applications to be designed while the database module implementations for the abstract interface are being designed.

Using Ada as the implementation language supports the concepts of abstraction, information hiding, and strong typing. All of these concepts are necessary to accomplish a design built from good software engineering principles. Systems built using good software engineering principles have more predictable behavior and better maintainability.

Recommendations

The following recommendations are made to apply the results of this research and to continue on with the work of this research:

- The developing organization should re-design the STEMdB to make it more like a DSS. They should use an object oriented approach which isolates the areas likely to change in the Knowledge Base subsystem part of the new design.
- The developing organization should incorporate the improvements to the present design presented in Chapter III to make the design more robust, maintainable and portable.
- The developing organization should make the STEMdB accessible to remote users and more portable by localizing the user interface dependencies and by designing the STEMdB in Ada using an abstract interface to communicate with an SQL database.

- Accomplish further research in the area of Ada to SQL bindings by implementing the abstract interface specification in this research. This implementation would require experience in both the SQL language, the SAME methodology, and the STEMdB application.

- Use the SAME description language to develop a DOD SAME compiler that would automate the abstract interface process and allow three expert domain engineers (The SQL implementation designer, the Abstract interface designer and the Application program designer) to concentrate on their area of expertise. The description language, SAMEDL, has already been defined for this approach. All that would be required is an understanding of how to develop an Ada compiler-like language.

Summary

The concept of the STEMdB is promising. The need for such a system exists and will continue to exist as long as CASE tools are required to help manage the complicated software development process. A sound, flexible design of a STEMdB system will serve the needs of the Air Force and DOD for many years to come.

Appendix A. Example STSC Listing of Software Characteristics and Qualities

This appendix provides an example of the Software Technology Support Center's accomplishments towards identifying CASE tool software characteristics and quality information in the Software Area of Interest of Requirements Analysis and Design. This information comes directly from [18:66-76].

B.1 Functional

The following sections identify the functional capabilities of Upper CASE tools. The organizational breakout is identical to the one presented in Section 2.2.2.1.

B.1.1 Information Capture

The information capture functionality area deals with what types of information the tool is capable of handling. This information can be captured in a number of different ways. The important idea is what type of information is captured, not how it is captured. Table B-1, Upper CASE Tool Information Types, lists the types of information that Upper CASE tools capture.

- System Function Descriptions
- Data Descriptions of System Functions Interfaces
- Data Descriptions of System Input/Output Device Interfaces
- System Logical Behavior
- System Timing Behavior
- Hardware/Software Context
- Software Architectural Structure
- Software Process Definitions
- Software Data Structures
- Software Process Control
- Software Process Concurrency
- Software Inter-Process Data Communication
- Software Inter-Process Synchronization

Table B-1. Upper CASE Tool Information Types

B.1.2 Methodology Support

Methodology is the process the tool user follows to systematically develop correct and complete work products. A number of methodologies exist for requirements analysis and software design. The important ones that have been automated are listed in Table B-2, Upper CASE Methodologies.

• Real-Time Structured Development	• PAMELA
• Structured Analysis	• ESML
• Structured Design	• ADARTS
• Hatley/Pirbhai Extensions	• PAISLey
• Object-Oriented Design	• VDM
• Ada-Based Object-Oriented Design	• Petri Nets
• Object-Oriented Analysis	• Statecharts
• Entity Relationship Modeling	• Axiomatic Specification

Table B-2. Upper CASE Methodologies

These methodologies require that various work products be created by the user. Since different methodologies can require the same work products, the products are listed separately in Table B-3, Upper CASE Tool Products.

• Data Flow Diagrams	• Ada Package Dependency Diagrams
• System Context Diagrams	• Structure Char.s
• Block Diagrams	• Flow Charts
• Control Flow Diagrams	• Screen and Report Diagrams
• Entity Relationship Diagrams	• User Tailored Diagrams
• State Transition Tables	• Object Oriented Diagrams
• Petri Net Diagrams	• Object Hierarchy or Tree Diagrams
• Architecture Diagrams	• Object Diagrams
• Object Interaction Diagrams	

Table B-3. Upper CASE Tool Products

B.1.3 Model Analysis

The model analysis functionality area captures the techniques the tool uses to analyze the inputs. These techniques can be static or dynamic. They are used to prove qualities about the input requirements or specifications such as completeness or consistency. They are also used to simulate the inputs at an early stage in the life cycle. The important techniques are listed in Table B-4, Upper CASE Analysis Techniques.

• Consistency Checking	• Behavior Analysis
• Completeness Checking	• Scenario-Based Analysis
• Data Normalization Analysis	• Exhaustive Model Analysis
• Man/machine Interface Analysis	

Table B-4. Upper CASE Analysis Techniques

B.1.4 Requirements Tracing

The requirements tracing functionality area captures the attributes associated with the tracing of requirements between software life cycle phases. Requirements tracing is important because it facilitates the management of inter life cycle dependencies. The important attributes are listed in Table B-5, Requirements Tracing Attributes.

- Extraction of Requirements from System and Software Documentation
- Inputs From Electronically Scanned Hard-Copy
- Multiple Requirements Baselines
- Tracing of System Requirements to Software Requirements
- Tracing of System Design Specifications to Software Requirements
- Tracing of Requirements to Software Design
- Tracing of Requirements to Source Code
- Tracing of Requirements to Software and System Test

Table B-5. Requirements Tracing Attributes

B.1.5 Data Repository

The data repository functionality area captures the attributes associated with the database the tool uses. Most tools use proprietary databases. The database model presented to the user, the user interface to the database, and the extent to which the database can represent software objects is critical to the database's overall functionality. The important attributes are listed in Table B-6, Upper CASE Data Repository Functional Attributes.

• Data Repository	• Contain Project Information
• Relational Database Type	• Contain Requirements Documents
• Object Database Type	• Contain Design Specifications
• Support both Text and Graphics	• Contain Source Code
• Query Capability	• Contain Test Descriptions & Procedures
• Access Control Capability	• Capacity Artificially Limited
• Concurrent Access to Entities	• Support Interactive Cross-Referencing

Table B-6. Upper CASE Data Repository Functional Attributes

B.1.6 Documentation

The documentation functionality area captures the attributes associated with the documentation the tool produces. The important attributes are listed in Table B-7, Upper CASE Documentation Functional Attributes.

• Support Graphics/Text Integration	• Automatic Generation of Documentation
• Completely Compile a Document	• Rapid Draft Hard-Copy
• On-Line Templates	• Interface to Other Document Generators
• 2167A Documentation Standard	• Desktop Publishing Interface

Table B-7. Upper CASE Documentation Functional Attributes

B.1.7 Data Import/Export

The data import/export functionality area captures the attributes associated with how easily the tool can exchange data with other tools including other tools in the tool vendor's tool set. The important attributes are listed in Table B-8, Upper CASE Data I/O Functional Attributes.

- Between Toolkit Components
- With Other Tools
- CAIS-A Interface Standards/Protocols Supported

Table B-8. Upper CASE Data I/O Functional Attributes

B.1.8 Reusability Support

The reusability functionality area captures the attributes associated with how the tool supports reuse. The one attribute in this area deals with support for library design components.

B.2 Quality

The following sections define and discuss the Upper CASE implications of the twelve quality attributes identified in the analysis phase.

B.2.1 Efficiency

Upper CASE tool efficiency is the amount of utilization of a resource on a problem, using the Upper CASE tool. The three resources that need to be assessed are processor (time to complete a task), memory (the secondary storage requirements to complete a task), and communication (I/O and network considerations for multi-processor systems and/or multiuser problems). For Upper CASE tools, efficiency is not absolutely expressed. Instead, it is expressed in terms of acceptable, barely acceptable, or unacceptable. Several problems covering a range of sizes from small to large across each of the resources need to be assessed. When the tool performs adequately for a specific problem with respect to a particular resource, its efficiency is acceptable for that problem size and that resource. Barely acceptable performance occurs when the performance is acceptable but there is no room for performance growth.

Efficiency as it applies to the products of Upper CASE tools is not important. This is because the products of these tools are paper reports. That the tools may support efficiency studies of their products (e.g., timing analysis of designs) is a matter of functionality and not quality.

B.2.2 Integrity

Integrity deals with either software security failures due to unauthorized access or the corruption of the database. As a policy, the tool users should lose confidence in the integrity of the database if unauthorized access is allowed. Database corruption may be caused by such actions as legal but partial and/or inconsistent operations and erroneous but allowed operations.

The integrity of the products of the tool is a non-issue. Accessibility to the products is usually governed by the operating system of the developmental machine and never by the tool itself. Once a product has been produced it is no longer part of the database and can no longer be corrupted.

B.2.3 Reliability

Reliability concerns software failures. Reliability is normally measured by direct testing and analysis of error reports. With commercial software, direct testing is not feasible and detailed error reports are not normally published. For Upper CASE tools, instead of directly measuring reliability, indicators such as maturity, published error reports, size of executable code, and errors uncovered during testing will be used.

Since the products of the Upper CASE tools are themselves intermediate products of the entire software development process, their reliability cannot be tested.

B.2.4 Survivability

Survivability deals with the ability of the software to perform even when portions of the system have failed. This issue is not usually important in the evaluation of Upper CASE tools because the greater issue of system availability is not critical in an office environment.

However, if the tool uses different hardware resources (i.e., networked workstations with a file server), the issue of how the tool handles hardware resource failure (i.e., file server shutdown) must be addressed.

Survivability is not an issue for the tool products because they are reports.

B.2.5 Usability

Usability is the extent to which resources required to acquire, install, learn, operate, prepare input for, and interpret output of the tool or the tool products are minimized. This attribute is probably the most important and critical quality attribute that Upper CASE tools are evaluated for. This is the quality attribute in which tool vendors differentiate themselves through such quality criteria as user interfaces, user documentation, and training.

The usability of the products of the tool is not an important quality issue. The ability to customize reports is addressed in the documentation portion of the functional capabilities of the tool.

B.2.6 Correctness

Correctness is the extent to which software design and implementation conform to specifications and standards. The correctness of a tool is evaluated in other portions of the evaluation framework, namely the functional capability area. The reliability quality attribute addresses known errors.

The correctness of the tool products is important. The generated products should conform to the specification captured by the tool.

B.2.7 Maintainability

Maintainability is the ease of effort for locating and fixing software failures within a specified time period. This attribute is not of importance to the tool user, instead the time and ability for the vendor to deliver software maintenance is important. The tool user is not concerned with the effort required to perform these actions. This time is addressed in the vendor information portion of the evaluation framework (under management concerns).

This attribute is of importance to the the user of the products of the tool, but not to the tool user. The tool products should possess the quality attributes of maintainability.

B.2.8 Verifiability (aka testability)

Verifiability deals with the design characteristics that facilitate the testing of the tool or the tool's products. The testing of the tool is important to the developers of the tool but not to the tool user (except that a well tested tool will have higher reliability, etc.).

The ability to test the tool's products is important in determining the quality of the tool. But for Upper CASE tools, testing is best addressed as a functional capability of the tool.

B.2.9 Expandability (aka flexibility)

Expandability is the ease in which current functions can be enhanced or new functions added. Flexibility is defined as the ease in which the software can be changed to meet other new requirements. Within the scope of evaluating Upper CASE tools and Upper CASE tool products, where the viewpoint is user implemented changes (not developer implemented changes), these attributes are dealt with in the reusability quality attribute.

B.2.10 Interoperability

Interoperability is the ability of separate systems to exchange database objects and their relationships without conversion. This is an important area, capturing if, how much, and how well the Upper CASE tool implements data exchange standards. This area is addressed in the Functional Capabilities portion of the evaluation framework. It is not an important quality attribute for either Upper CASE tools or their products.

B.2.11 Reusability

Reusability is the extent to which a component can be adapted for use in another application. Within the scope of evaluating Upper CASE tools, reusability deals with how easily the tool can be used for other projects.

The issue of reusability of the products of the CASE tool is dealt with in the functional capabilities portion of the evaluation framework.

B.2.12 Transportability (aka Portability)

Transportability is the ability of a software item to be installed in a different environment without change in functionality. Within the scope of evaluating Upper CASE tools, it deals with how many platforms and operating systems the tool works with. This area is addressed in the portion of the evaluation framework dealing with operational constraints.

This is a non-issue with Upper CASE tool products since, by their very nature, they are reports not associated with any particular environment.

Appendix B. SAME Package Listings

This appendix provides a listing of all SAME Ada Package specifications and bodies that were used in this research. The listings are directly out of Graham's technical report (CMU/SEI-89-TR-16) in [17:143-248].

SAME Standard Package Listings

C.1 Introduction

This appendix contains the source code of the SAME standard packages. This code will be available in machine-readable form from the SEI for a limited time. Please read the copyright notice in the next section. A copy of this notice appears in each file of the machine-readable distribution.

Every procedure and function declaration in these packages is followed by a pragma INLINE which has been "commented out." The explanation for this is as follows. Almost all of the procedures and functions in these packages are extremely small. Many consist of a single If or return statement. Therefore they are excellent candidates for procedure inlining which will decrease their runtime cost by the overhead of a procedure call. Experience in using this code with various compilers has shown that this degree of inlining tends to uncover compiler errors and produce inexplicable timings. The safest approach, that of not using inlining at all, has be chosen for the code as distributed. The installer is urged to experiment with the inlining of this code. Some experiments have shown a tenfold speedup due to inlining (whereas other experiments, on other compilers and machine architectures, showed marginal slowdown due to inlining). Recall that inlining will usually make the resulting object module larger.

C.4 SQL_Standard Specification

```
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 Decimal is to be determined;
    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;

--    csp is an implementor-defined package and cst is an
--    implementor-defined character type. bs, ts, bi, ti, dr, dd, bsc,
--    and tsc are implementor defined integral values. t is int or
--    smallint corresponding to an implementor-defined <exact
--    numeric type> of indicator parameters.

end sql_standard;
```

C.5 SQL_Communications_Pkg Specification

```
with SQL_Char_Pkg; use SQL_Char_Pkg;
with SQL_Standard; use SQL_Standard;
package SQL_Communications_Pkg is

    -- This is an example of the package, providing minimal functionality.
    -- This package may be tailored to the needs of a given platform.

    SQL_Database_Error : exception;

    SQLCODE : SQLCODE_TYPE;

    -- Parameterless function returning an error message of type
    --    SQL_Char_Not_Null.
    -- The error message is the descriptive string associated with
    --    the most recent database error. It is produced by a
    --    DBMS specific function.

    function SQL_Database_Error_Message return SQL_Char_Not_Null;

end SQL_Communications_Pkg;
```

C.6 SQL_Communications_Pkg Body

```
-- SQL_Communications_Pkg is a "platform-specific" package
--   within the SAME
-- this particular version of the package was developed for
--   a platform consisting of the Verdix (Version 5.41) Ada compiler
--   and INGRES (Version 5.0) running on a Vax Station
```

```
with system; use system;
with SQL_System; use SQL_System;
with ingres_c_support; use ingres_c_support;
-- ingres_c_support contains functions Add_Null and Strip_Null
-- which are used to convert between 'c' format strings and
-- Ada format strings. It is not included in the SAME standard packages.
package body SQL_Communications_Pkg is

function SQL_Database_Error_Message return SQL_Char_Not_Null is

    Message_Buffer : SQL_Char_Not_Null (1..MAXERRLEN);

    Len : integer := MAXERRLEN;

    procedure geterrmsg (Message : in Address;
                         Length  : in Address);

    pragma interface(C, geterrmsg, "_sqlerrmsg");

begin
    geterrmsg (Message_Buffer'Address, Len'Address);

    -- the assumption here is that no error will occur when
    --   retrieving the error message from the database

    return strip_null(Message_Buffer);

end SQL_Database_Error_Message;


end SQL_Communications_Pkg;
```

C.7 SQL_Exceptions Specification

```
package SQL_exceptions
     is

   Null_Value_Error : exception;

end SQL_exceptions;
```

C.8 SQL_Boolean_Pkg Specification

```
package SQL_Boolean_Pkg
     is

   type Boolean_with_Unknown is (FALSE, UNKNOWN, TRUE);

   ---- Three valued Logic operations --
   ---  three-val X three-val => three-val --
   function "not"  (Left : Boolean_with_Unknown)
                    return Boolean_with_Unknown;
   -- pragma INLINE ("not");
   function "and" (Left, Right : Boolean_with_Unknown)
                    return Boolean_with_Unknown;
   -- pragma INLINE ("and");
   function "or" (Left, Right : Boolean_with_Unknown)
                    return Boolean_with_Unknown;
```

C.2 Copyright Notice

C.3 SQL_System Specification

```
-- SQL_System is a "platform-specific" package
--   within the SAME

package SQL_System is

    -- MAXCHRLEN is the  length of the longest character string
    -- which the DBMS will store.
    -- It serves as the upper bound on SQL_Char_Pkg
    --   subtypes SQL_Char_Length and SQL_Unpadded_Length.
    -- SQL_Char_Length is a subtype of Natural with a lower bound
    --   of 1.
    -- SQL_Unpadded_Length is a subtype of Natural with a lower
    --   bound of 0.

    MAXCHRLEN : constant integer := str_length; -- replace

    -- MAXERRLEN is the maximum length of the error message
    --   string returned from DBMS specific error message routine

    MAXERRLEN : constant integer := msg_length;  -- replace

end SQL_System;
```

```
--   for the SQL_Interval type
function "+"(Right : SQL_Interval) return SQL_Interval;
function "-"(Right : SQL_Interval) return SQL_Interval;
function "abs"(Right : SQL_Interval) return SQL_Interval;
-- pragma INLINE ("abs");

-- the following functions implement three valued
--   arithmetic
-- if either input to any of these functions is null
--   the function returns the null value; otherwise
--   they perform the indicated operation
-- these functions raise no exceptions
function "+"(Left, Right : SQL_Interval) return SQL_Interval;
function Plus(Left : SQL_Interval; Right : SQL_Date) return SQL_Date;
function Plus(Left : SQL_Date; Right : SQL_Interval) return SQL_Date;
-- pragma INLINE ("+");
function "-"(Left, Right : SQL_Interval) return SQL_Interval;
function Minus(Left, Right : SQL_Date) return SQL_Interval;
function Minus(Left : SQL_Date; Right : SQL_Interval) return SQL_Date;
-- pragma INLINE ("-");
function "*"(Left : SQL_Interval; Right : integer) return SQL_Interval;
-- pragma INLINE ("*");
function "/"(Left : SQL_Interval; Right : integer) return SQL_Interval;
-- pragma INLINE ("/");

      -- Logical Operations --
      -- type X type => Boolean_with_unknown --
-- these functions implement three valued logic
-- if either input is the null value, the functions
--   return the truth value UNKNOWN; otherwise they
--   perform the indicated comparison.
-- these functions raise no exceptions
function Equals (Left, Right : SQL_Date) return Boolean_with_Unknown;
function Equals (Left, Right : SQL_Interval) return Boolean_with_Unknown;
-- pragma INLINE (Equals);
function Not_Equals (Left, Right : SQL_Date)
                          return Boolean_with_Unknown;
function Not_Equals (Left, Right : SQL_Interval)
                          return Boolean_with_Unknown;
-- pragma INLINE (Not_Equals);
function "<" (Left, Right : SQL_Date) return Boolean_with_Unknown;
function "<" (Left, Right : SQL_Interval) return Boolean_with_Unknown;
-- pragma INLINE ("<");
function ">" (Left, Right : SQL_Date) return Boolean_with_Unknown;
function ">" (Left, Right : SQL_Interval) return Boolean_with_Unknown;
-- pragma INLINE (">");
function "<=" (Left, Right : SQL_Date) return Boolean_with_Unknown;
function "<=" (Left, Right : SQL_Interval) return Boolean_with_Unknown;
-- pragma INLINE ("<=");
function ">=" (Left, Right : SQL_Date) return Boolean_with_Unknown;
function ">=" (Left, Right : SQL_Interval) return Boolean_with_Unknown;
-- pragma INLINE (">=");

      -- type => boolean --
function Is_Null(Value : SQL_Date) return Boolean;
function Is_Null(Value : SQL_Interval) return Boolean;
-- pragma INLINE (Is_Null);
function Not_Null(Value : SQL_Date) return Boolean;
function Not_Null(Value : SQL_Interval) return Boolean;
-- pragma INLINE (Not_Null);
function Is_Year_Month(Value : SQL_Interval) return Boolean;
-- pragma INLINE(Is_Year_Month);
function Is_Day_Time(Value : SQL_Interval) return Boolean;
```

```
type SQL_Datetime_Field is (year, month, day,
                            hour, minute, second, fraction);
type SQL_Date_Not_Null is new SQL_Char_Not_Null;
type SQL_Date(From       : SQL_Datetime_Field;
              To         : SQL_Datetime_Field;
              Fractional : precision) is limited private;
type SQL_Interval(From       : SQL_Datetime_Field;
                  Leading    : precision;
                  To         : SQL_Datetime_Field;
                  Fractional : precision) is limited private;

function Null_SQL_Date return SQL_Date;
-- pragma INLINE (Null_SQL_Date);

function Null_SQL_Interval return SQL_Interval;
-- pragma INLINE (Null_SQL_Interval);

-- these functions return the not-null portion of the null-bearing type
function Without_Null_Base(Value : SQL_Date) return SQL_Date_Not_Null;
function Without_Null_Base(Value : SQL_Interval) return SQL_Date_Not_Null;
-- pragma INLINE (Without_Null_Base);

-- this function returns an object of the standard.duration type, after
--    converting to it from the input object of type SQL_Interval
function To_Duration (Value : SQL_Interval) return duration;
-- pragma INLINE (To_Duration);

-- this function returns an object of the calendar.time type, after
--    converting to it from the input object of type SQL_Date
function To_Time (Value : SQL_Date) return time;
-- pragma INLINE (To_Time);

-- these procedures parse the input of type SQL_Date_Not_Null, and assign
--    the datetime and interval field values to the objects of type
--    SQL_Date and SQL_Interval, using discriminants that it determines are
--    the correct ones for the object.  If these discriminants differ from
--    the ones supplied in the abstract domain for the object when it was
--    declared, a constraint_error will be raised.
procedure Parse_and_Assign_Base(Left: in out SQL_Date;
                                Right : SQL_Date_Not_Null);
procedure Parse_and_Assign_Base(Left: in out SQL_Interval;
                                Right :SQL_Date_Not_Null);
-- pragma INLINE (Parse_and_Assign);

-- this function accepts input of type standard.duration, and
--    returns an object of type SQL_Interval whose not-null portion
--    has the correct SQL "interval" value specification format,
--    (FROM => day, LEADING => 2, TO => fraction, FRACTIONAL => 3)
function To_SQL_Interval (Value : duration) return SQL_Interval;
-- pragma INLINE (To_SQL_Interval);

-- this function accepts input of type standard.time, and
--    returns an object of type SQL_Date whose not-null portion
--    has the correct SQL "datetime" value specification format
function To_SQL_Date (Value : time) return SQL_Date;
-- pragma INLINE (To_SQL_Date);

-- the assign procedure assigns Right to Left
procedure Assign(Left : in out SQL_Date; Right : SQL_Date);
procedure Assign(Left : in out SQL_Interval; Right : SQL_Interval);
-- pragma INLINE (Assign);

-- the following three functions implement unary "+", "-", "abs"
```

```
            return With_Null(SQL_Enumeration_Not_Null'Value(
                                 To_String(Value)));
        end if;
    end Value;

end SQL_Enumeration_Pkg;
```

C.28 SQL_Database_Error_Pkg Specification

```
package SQL_Database_Error_Pkg is

    -- The following procedure must be present in every version of
    -- SQL_Database_Error_Pkg. It's purpose is to perform standard
    -- processing of unexpected exceptional conditions. It should not
    -- attempt error recover.

    procedure Process_Database_Error;

end SQL_Database_Error_Pkg;
```

C.29 SQL_Database_Error_Pkg Body

```
with Text_IO, SQL_Communications_Pkg, SQL_Base_Types_Pkg;
use Text_IO, SQL_Communications_Pkg, SQL_Base_Types_Pkg;
package body SQL_Database_Error_Pkg is

    procedure Process_Database_Error is

    begin

        -- Procedure Process_Database_Error is called in response
        --   to an unexpected database exception (an error incident).
        --  The procedure may be modified per
        --   the needs of the Abstract Interface developer

        -- This is a minimal implementation.
        --   Get a descriptive error message from the DBMS
        --    (through the package SQL_Communications_Pkg)
        --   and display it on standard output.

        put_line (To_String(SQL_Char_Not_Null(SQL_Database_Error_Message)));

    end Process_Database_Error;

end SQL_Database_Error_Pkg;
```

C.30 SQL_Date_Pkg Specification

```
with SQL_Standard;
with Calendar; use Calendar;
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Char_Pkg; use SQL_Char_Pkg;
package SQL_Date_Pkg
    is

    type precision is range 0..10;
```

```
          Is_Null: Boolean := true;
          Value: SQL_Enumeration_Not_Null;
      end record;

end SQL_Enumeration_Pkg;
```

C.27 SQL_Enumeration_Pkg Body

```
With SQL_Exceptions;
package body SQL_Enumeration_Pkg
        is

    Null_Value_Error : exception renames SQL_Exceptions.Null_Value_Error;

    function Null_SQL_Enumeration return SQL_Enumeration is
        Null_Holder : SQL_Enumeration;
    begin
        return Null_Holder;
    end Null_SQL_Enumeration;

    function Without_Null(Value : in SQL_Enumeration)
      return SQL_Enumeration_Not_Null is
    begin
        if Value.Is_Null then
            raise Null_Value_Error;
        else
            return Value.Value;
        end if;
    end Without_Null;

    function With_Null(Value : in SQL_Enumeration_Not_Null)
      return SQL_Enumeration is
    begin
        return (Is_Null => false,
                Value   => Value);
    end With_Null;

    procedure Assign (Left  : in out SQL_Enumeration;
                      Right : in SQL_Enumeration) is
    begin
        Left := Right;
    end Assign;

    function Equals (Left, Right : SQL_Enumeration)
        return Boolean_With_Unknown is
    begin
        if Left.Is_Null or else Right.Is_Null then
            return Unknown;
        elsif Left.Value = Right.Value then
            return True;
        else
            return False;
        end if;
    end Equals;

    function Not_Equals (Left, Right : SQL_Enumeration)
        return Boolean_With_Unknown is
    begin
        if Left.Is_Null or else Right.Is_Null then
            return Unknown;
        -- these functions raise no exceptions
        function Equals (Left, Right : SQL_Enumeration)
            return Boolean_with_Unknown;
        function Not_Equals (Left, Right : SQL_Enumeration)
            return Boolean_with_Unknown;
        -- pragma INLINE (Not_Equals);
        function "<" (Left, Right : SQL_Enumeration) return Boolean_with_Unknown;
        function ">" (Left, Right : SQL_Enumeration) return Boolean_with_Unknown;
        function "<=" (Left, Right : SQL_Enumeration) return Boolean_with_Unknown;
        function ">=" (Left, Right : SQL_Enumeration) return Boolean_with_Unknown;

              -- type => boolean --
        function Is_Null (Value : SQL_Enumeration) return Boolean;
        -- pragma INLINE (Is_Null);
        function Not_Null (Value : SQL_Enumeration) return Boolean;
        -- pragma INLINE (Not_Null);
        function "=" (Left, Right : SQL_Enumeration) return Boolean;
        -- pragma INLINE ("=");
        function "<" (Left, Right : SQL_Enumeration) return Boolean;
        -- pragma INLINE ("<");
        function ">" (Left, Right : SQL_Enumeration) return Boolean;
        -- pragma INLINE (">");
        function "<=" (Left, Right : SQL_Enumeration) return Boolean;
        -- pragma INLINE ("<=");
        function ">=" (Left, Right : SQL_Enumeration) return Boolean;
        -- pragma INLINE (">=");


        -- the following six functions mimic the
        --    'Pred, 'Succ, 'Image, 'Pos, 'Val, and 'Value
        --    attributes of the SQL_Enumeration_Not_Null type, passed
        --    in, for the associated SQL_Enumeration (null) type
        -- they all raise the Null_Value_Error exception if a null
        --    value is passed in
        -- Pred raises the Constraint_Error exception if the value
        --    passed in is equal to SQL_Enumeration_Not_Null'Last
        -- Succ raises the Constraint_Error exception if the value
        --    passed in is equal to SQL_Enumeration_Not_Null'First
        -- Val raises the Constraint_Error exception if the value passed
        --    in is not in the range P'POS(P'FIRST)..P'POS(P'LAST) for type P
        -- Value raises the Constraint_Error exception if the sequence of
        --    characters passed in does not have the syntax of an enumeration
        --    literal for the instantiated enumeration type
        function Pred (Value : in SQL_Enumeration) return SQL_Enumeration;
        -- pragma INLINE (Pred);
        function Succ (Value : in SQL_Enumeration) return SQL_Enumeration;
        -- pragma INLINE (Succ);
        function Pos (Value : in SQL_Enumeration) return Integer;
        -- pragma INLINE (Pos);
        function Image (Value : in SQL_Enumeration) return SQL_Char;
        function Image (Value : in SQL_Enumeration_Not_Null)
            return SQL_Char_Not_Null;
        -- pragma INLINE (Image);
        function Val (Value : in Integer) return SQL_Enumeration;
        -- pragma INLINE (Val);
        function Value (Value : in SQL_Char) return SQL_Enumeration;
        function Value (Value : in SQL_Char_Not_Null)
            return SQL_Enumeration_Not_Null;
        -- pragma INLINE (Value);

    private

        type SQL_Enumeration is record
    -- pragma INLINE ("<=");
    function ">=" (Left, Right : SQL_Char) return Boolean_with_Unknown;
    -- pragma INLINE (">=");

          -- type => boolean --
    function Is_Null(Value : SQL_Char) return Boolean;
    -- pragma INLINE (Is_Null);
    function Not_Null(Value : SQL_Char) return Boolean;
    -- pragma INLINE (Not_Null);

    -- These functions of class type => boolean
    -- equate UNKNOWN with FALSE. That is, they return TRUE
    -- only when the function returns TRUE. UNKNOWN and FALSE
    -- are mapped to FALSE.
    function "=" (Left, Right : SQL_Char) return Boolean;
    -- pragma INLINE ("=");
    function "<" (Left, Right : SQL_Char) return Boolean;
    -- pragma INLINE ("<");
    function ">" (Left, Right : SQL_Char) return Boolean;
    -- pragma INLINE (">");
    function "<=" (Left, Right : SQL_Char) return Boolean;
    -- pragma INLINE ("<=");
    function ">=" (Left, Right : SQL_Char) return Boolean;
    -- pragma INLINE (">=");


    -- the purpose of the following generic is to generate
    --    conversion functions between a type derived from
    --    SQL_Char_Not_Null, which are effectively Ada
    --    strings and a type derived from SQL_Char, which
    --    mimic the behaviour of SQL strings.
    -- the subprogram formals are meant to default; that is,
    --    this generic should be instantiated in the scope
    --    of an use clause for SQL_Char_Pkg.
generic
    type With_Null_Type is limited private;
    type Without_Null_Type is array (positive range <>)
        of sql_standard.Character_type;
    with function With_Null_Base (Value: SQL_Char_Not_Null)
        return With_Null_Type is <>;
    with function Without_Null_Base (Value: With_Null_Type)
        return SQL_Char_Not_Null is <>;
    with function Without_Null_Unpadded_Base (Value: With_Null_Type)
        return SQL_Char_Not_Null is <>;
package SQL_Char_Ops is
    function With_Null (Value : Without_Null_Type)
        return With_Null_Type;
    -- pragma INLINE (With_Null);
    function Without_Null (Value : With_Null_Type)
        return Without_Null_Type;
    -- pragma INLINE (Without_Null);
    function Without_Null_Unpadded (Value : With_Null_Type)
        return Without_Null_Type;
    -- pragma INLINE (Without_Null_Unpadded);
end SQL_Char_Ops;

private

    type SQL_Char(Length : SQL_Char_Length) is record
        Is_Null: Boolean := true;
        Unpadded_Length: SQL_Unpadded_Length;
        Text: SQL_Char_Not_Null(1 .. Length);
    end record;

    end SQL_Char_Pkg;
```

C.24 Subunit To_String

```
-- assuming an ascii host character set
-- that is SQL_Standard.Character_Type is Standard.Character
separate (SQL_Char_Pkg)
function To_String (Value : SQL_Char_Not_Null)
    return String is
begin
    return (String(Value));
end To_String;
```

C.25 Subunit To_SQL_Char_Not_Null

```
-- assuming an ascii host character set
-- that is SQL_Standard.Character_Type is Standard.Character
separate (SQL_Char_Pkg)
function To_SQL_Char_Not_Null (Value : String)
    return SQL_Char_Not_Null is
begin
    return (SQL_Char_Not_Null(Value));
end To_SQL_Char_Not_Null;
```

C.26 SQL_Enumeration_Pkg Specification

```
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Char_Pkg; use SQL_Char_Pkg;
generic
    type SQL_Enumeration_Not_Null is (<>);
package SQL_Enumeration_Pkg
      is

        ---- Possibly Null Enumeration ----
    type SQL_Enumeration is limited private;

    function Null_SQL_Enumeration return SQL_Enumeration;
    -- pragma INLINE (Null_SQL_Enumeration);

    -- this pair of functions convert between the
    --    null-bearing and non-null-bearing types.
    function Without_Null(Value : in SQL_Enumeration)
      return SQL_Enumeration_Not_Null;
    -- pragma INLINE (Without_Null);
    -- With_Null raises Null_Value_Error if the input
    --    value is null
    function With_Null(Value : in SQL_Enumeration_Not_Null)
      return SQL_Enumeration;
    -- pragma INLINE (With_Null);

    procedure Assign (
        Left : in out SQL_Enumeration; Right : in SQL_Enumeration);
    -- pragma INLINE (Assign);

          -- Logical Operations --
          -- type X type => Boolean_with_unknown --
    -- these functions implement three valued logic
    -- if either input is the null value, the functions
    --    return the truth value UNKNOWN; otherwise they
    --    perform the indicated comparison.
```

```
function To_String (Value : SQL_Char)
    return String;
function To_Unpadded_String (Value : SQL_Char_Not_Null)
    return String;
function To_Unpadded_String (Value : SQL_Char)
    return String;
-- pragma INLINE (To_Unpadded_String);
-- this INLINE works for BOTH functions!!
function To_SQL_Char_Not_Null (Value : String)
    return SQL_Char_Not_Null;
function To_SQL_Char (Value : String)
    return SQL_Char;
-- pragma INLINE (To_SQL_Char);

function Unpadded_Length (Value : SQL_Char)
    return SQL_Unpadded_Length;
-- pragma INLINE (Unpadded_Length);

procedure Assign(
  Left : out SQL_Char;
  Right : SQL_Char
);
-- pragma INLINE (Assign);

-- Substring(x,k,m) returns the substring of x starting
--    at position k (relative to 1) with length m.
-- returns null value if x is null
-- raises constraint_error if Start < 1 or Length < 1 or
--    Start + Length - 1 > x.Length
function Substring (Value : SQL_Char;
                    Start, Length : SQL_Char_Length)
        return SQL_Char;
-- pragma INLINE (Substring);

-- "&" returns null if either parameter is null;
--    otherwise performs concatenation in the usual way,
--    preserving all blanks.
-- may raise constraint_error implicitly if result is
--    too large (i.e., greater than SQL_Char_Length'Last
function "&" (Left, Right : SQL_Char)
    return SQL_Char;
-- pragma INLINE ("&");

      -- Logical Operations --
      -- type X type => Boolean_with_unknown --
-- the comparison operators return the boolean value
--    UNKNOWN if either parameter is null; otherwise,
--    the comparison is done in accordance with
--    ANSI X3.135-1986 para 5.11 general rule 5; that is,
--    the shorter of the two string parameters is
--    effectively padded with blanks to be the length of
--    the longer string and a standard Ada comparison is
--    then made
function Equals (Left, Right : SQL_Char) return Boolean_with_Unknown;
-- pragma INLINE (Equals);
function Not_Equals (Left, Right : SQL_Char)
                        return Boolean_with_Unknown;
-- pragma INLINE (Not_Equals);
function "<" (Left, Right : SQL_Char) return Boolean_with_Unknown;
-- pragma INLINE ("<");
function ">" (Left, Right : SQL_Char) return Boolean_with_Unknown;
-- pragma INLINE (">");
function "<=" (Left, Right : SQL_Char) return Boolean_with_Unknown;
```

```
UPPER    DC     PL16'2147483648'
POSCON   DC     X'C0000000'
NEGCON   DC     X'D0000000'
POSZCON  DC     X'F0000000'
ZERO     DC     F'0'
ONE      DC     F'1'
SXTEEN   DC     F'16'
THRTYTWO DC     F'32'
MULV1A   MP     0(0,2),0(0,3)
MULV2A   MP     0(0,2),0(0,4)
DIVISN   DP     0(0,2),0(0,4)
         END    ADASUP
```

C.22 SQL_Char_Pkg Specification

```
with SQL_System; use SQL_System;
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Standard;

package SQL_Char_Pkg
      is

    subtype SQL_Char_Length is natural
        range 1 .. MAXCHRLEN;
    subtype SQL_Unpadded_Length is natural
        range 0 .. MAXCHRLEN;

    type SQL_Char_Not_Null is new SQL_Standard.Char;

    type SQL_Char(Length : SQL_Char_Length) is limited private;

    function Null_SQL_Char return SQL_Char;
    -- pragma INLINE (Null_SQL_Char);

    -- the next three functions convert between
    --   null-bearing and non null-bearing-types
    -- Without_Null_Base and With_Null_Base are
    --   inverses (mod. null values)
    -- see also SQL_Char_Ops generic package below
    function With_Null_Base(Value : SQL_Char_Not_Null)
      return SQL_Char;
    -- pragma INLINE (With_Null_Base);
    -- Without_Null_Base and Without_Null_Base_Unpadded raise
    --   null_value_error on the null input
    function Without_Null_Base(Value : SQL_Char) return SQL_Char_Not_Null;
    -- pragma INLINE (Without_Null_Base);
    -- Without_Null_Unpadded_Base removes trailing blanks from
    --   the input
    function Without_Null_Unpadded_Base(Value : SQL_Char)
      return SQL_Char_Not_Null;
    -- pragma INLINE (Without_Null_Unpadded_Base);
    -- axiom: unpadded_Length(x) =
    --   Without_Null_Unpadded_Base(x)'Length
    -- both functions raise null_value_error if x is null

    -- the next six functions convert between Standard.String
    --   types and the SQL_Char and SQL_Char_Not_Null types
    function To_String (Value : SQL_Char_Not_Null)
        return String;
```

```
     -- type Digit is picked to be an integer type with a range
     --    that will force the Ada compiler to pick a
     --    pre-defined integer type from package Standard.

     type Digit is range -(2**7)..(2**7)-1;

     -- the following object is declared so that the true size
     --    (in actual number of bits allocated) is assigned to the
     --    "size" object, rather than the number of bits used of
     --    those which are allocated.  In other words, using 'size
     --    on the type Digit yields 4 bits (number bits used),
     --    whereas using the 'size on "object" (of type Digit) yields
     --    8 bits (number bits allocated)

     object : Digit;

     -- size is the number of bits used by each object of type Digit
     -- it is used in the calculation of MAX_SIZE (below)

     size : constant integer := object'size;

     -- MAX_SIZE is the number of array positions needed for the
     --    Max_Decimal type below
     -- since each BCD digit can fit into 4 bits of storage, the
     --    total number of bits can be calculated by MAX_DIGITS * 4;
     -- this result is divided by the number of bits that an object
     --    of type Digit will comprise, which yields the number of
     --    array positions needed for the BCD number
     -- the result is incremented by one to accomodate the sign

     MAX_SIZE : constant integer := ((4 * (MAX_DIGITS)) / size) + 1;

     -- Max_Decimal is the array type definition used by the
     --    SQL_Decimal_Not_Null type definition (below) to allocate maximum
     --    storage for its BCD value

     type Max_Decimal is array (1..MAX_SIZE) of Digit;

     -- SQL_Decimal_Not_Null is the Ada BCD type.  It is comprised of a BCD
     --    value which resides in an object which reserves maximum
     --    space for BCD values, and a scale which indicates how
     --    many digits exist to the right of the decimal point in the
     --    BCD value

     type SQL_Decimal_Not_Null (scale : decimal_digits := 0) is record
         Value : Max_Decimal;
     end record;

     type SQL_Decimal_Not_Null2 (scale : decimal_digits := 0) is record
         Value : Max_Decimal;
     end record;

     type SQL_Decimal(scale : decimal_digits) is record
         Is_Null : boolean := true;
         Value   : SQL_Decimal_Not_Null(scale);
     end record;

end SQL_Decimal_Pkg;
```

```
generic
    type With_Null_Type(scale : decimal_digits) is limited private;
    type Without_Null_Type(scale : decimal_digits) is limited private;
    in_scale          : decimal_digits := 0;
    first_sign        : Sign_Character := '-';
    first_integral    : Numeric_String :=
                        (1..decimal_digits'last-in_scale => '9');
    first_fractional  : Numeric_String :=
                        (1..in_scale => '9');
    last_sign         : Sign_Character := '+';
    last_integral     : Numeric_String :=
                        (1..decimal_digits'last-in_scale => '9');
    last_fractional   : Numeric_String :=
                        (1..in_scale => '9');
    with function Is_In_Base (Right : Without_Null_Type;
                              Lower, Upper : SQL_Decimal_Not_Null2)
        return boolean is <>;
    with function Is_In_Base (Right : With_Null_Type;
                              Lower, Upper : SQL_Decimal_Not_Null2)
        return boolean is <>;
    with procedure Assign_with_check
                               (Left  : in out Without_Null_Type;
                                Right : Without_Null_Type;
                                Lower, Upper : SQL_Decimal_Not_Null2)
                                is <>;
    with procedure Assign_with_check
                               (Left  : in out With_Null_Type;
                                Right : With_Null_Type;
                                Lower, Upper : SQL_Decimal_Not_Null2)
                                is <>;
    with function To_SQL_Decimal_Not_Null2 (Value : Without_Null_Type)
        return SQL_Decimal_Not_Null2 is <>;
    with function To_SQL_Decimal_Not_Null2 (Value : With_Null_Type)
        return SQL_Decimal_Not_Null2 is <>;
    with function To_SQL_Decimal_Not_Null (Value : SQL_Decimal_Not_Null2)
        return Without_Null_Type is <>;
    with function To_SQL_Decimal (Value : SQL_Decimal_Not_Null2)
        return With_Null_Type is <>;
package SQL_Decimal_Ops is
    procedure Assign (Left  : in out Without_Null_Type;
                      Right : Without_Null_Type);
    procedure Assign (Left  : in out With_Null_Type;
                      Right : With_Null_Type);
    -- pragma INLINE(Assign);
    function Is_In(Right : Without_Null_Type)
        return boolean;
    function Is_In(Right : With_Null_Type)
        return boolean;
    -- pragma INLINE(Is_In);
    function With_Null (Value : Without_Null_Type)
        return With_Null_Type;
    -- pragma INLINE(With_Null);
    function Without_Null (Value : With_Null_Type)
        return Without_Null_Type;
    -- pragma INLINE(Without_Null_Type);
end SQL_Decimal_Ops;

private

        -- The requirement here is to provide
        -- at least enough space for the machine representation of the
        -- SQL_Decimal_Not_Null operands.
```

```
function To_SQL_Double_Precision_Not_Null (Right : SQL_Decimal)
    return SQL_Double_Precision_Not_Null;
-- pragma INLINE(To_SQL_Double_Precision_Not_Null);
function To_SQL_Double_Precision (Right : SQL_Decimal)
    return SQL_Double_Precision;
-- pragma INLINE(To_SQL_Double_Precision);

-- The following functions convert from Decimal to String:
function To_String  (Right : SQL_Decimal_Not_Null) return string;
function To_String  (Right : SQL_Decimal) return string;
-- pragma INLINE(To_String);
function To_SQL_Char_Not_Null (Right : SQL_Decimal_Not_Null)
    return SQL_Char_Not_Null;
function To_SQL_Char_Not_Null (Right : SQL_Decimal)
    return SQL_Char_Not_Null;
-- pragma INLINE(To_SQL_Char_Not_Null);
function To_SQL_Char (Right : SQL_Decimal) return SQL_Char;
-- pragma INLINE(To_SQL_Char);

-- the following functions return the length of the string
--    value returned by the "To_String" function
function Width      (Right : SQL_Decimal_Not_Null) return integer;
-- The following function raises the Null_Value_Error exception
--    on the null input
function Width      (Right : SQL_Decimal) return integer;
-- pragma INLINE(Width);

    -- The following functions implement some of the Ada Attributes
    -- of the BCD type

-- The number of BCD digits before the decimal point for the
-- type of the given object:
function Integral_Digits (Right : SQL_Decimal_Not_Null) return decimal_digits;
function Integral_Digits (Right : SQL_Decimal) return decimal_digits;
-- pragma INLINE(Integral_Digits);

-- The number of BCD digits after the decimal point for the
-- type of the given object:
function Scale (Right : SQL_Decimal_Not_Null) return decimal_digits;
function Scale (Right : SQL_Decimal) return decimal_digits;
-- pragma INLINE(Scale);

-- The actual number of BCD digits before the decimal point for
-- a given object of a given type:
function Fore (Right : SQL_Decimal_Not_Null) return positive;
-- The following function raises the Null_Value_Error on the null input
function Fore (Right : SQL_Decimal) return positive;
-- pragma INLINE(Fore);

-- The number of BCD digits after the decimal point for a
-- given object of a given type:
function Aft (Right : SQL_Decimal_Not_Null) return positive;
-- The following function raises the Null_Value_Error on the null input
function Aft (Right : SQL_Decimal) return positive;
-- pragma INLINE(Aft);

function Machine_Rounds (Right : SQL_Decimal_Not_Null) return boolean;
function Machine_Rounds (Right : SQL_Decimal) return boolean;
-- pragma INLINE(Machine_Rounds);

function Machine_Overflows (Right : SQL_Decimal_Not_Null) return boolean;
function Machine_Overflows (Right : SQL_Decimal) return boolean;
-- pragma INLINE(Machine_Overflows);
```

```
    return SQL_Decimal;
-- pragma INLINE("*");
function "/"   (Left : SQL_Decimal_Not_Null; Right : SQL_Int_Not_Null)
    return SQL_Decimal_Not_Null;
function "/"   (Left : SQL_Decimal; Right : SQL_Int_Not_Null)
    return SQL_Decimal;
function "/"   (Left : SQL_Decimal; Right : SQL_Int)
    return SQL_Decimal;
-- pragma INLINE("/");

-- The following functions convert to SQL_Decimal_Not_Null;
function To_SQL_Decimal_Not_Null (Right : SQL_Int_Not_Null)
    return SQL_Decimal_Not_Null;
-- the following function raises Constraint_Error
--     if the SQL_Double_Precision_Not_Null value is too large
--     to be represented in BCD format
function To_SQL_Decimal_Not_Null (Right : SQL_Double_Precision_Not_Null)
    return SQL_Decimal_Not_Null;
-- the following function raises Constraint_Error
--     if there are more than MAX_DIGITS number of digits;
--     if there are two or more decimal points;
--     if there are two or more sign designations;
--     if there exists a character other than '0'..'9' or '.'
--        or '+','-', ' ' for the sign
--     if the order of the characters is anything other than
--        sign designation followed by the number
function To_SQL_Decimal_Not_Null (Right : SQL_Char_Not_Null)
    return SQL_Decimal_Not_Null;
-- pragma INLINE(To_SQL_Decimal_Not_Null);

-- The following functions convert to SQL_Decimal;
function To_SQL_Decimal (Right : SQL_Int_Not_Null) return SQL_Decimal;
function To_SQL_Decimal (Right : SQL_Int) return SQL_Decimal;
-- the following two functions raise Constraint_Error
--     if the SQL_Double_Precision_Not_Null value is too large
--     to be represented in BCD format
function To_SQL_Decimal (Right : SQL_Double_Precision_Not_Null)
    return SQL_Decimal;
function To_SQL_Decimal (Right : SQL_Double_Precision) return SQL_Decimal;
-- the following two functions raise Constraint_Error
--     if there are more than MAX_DIGITS number of digits;
--     if there are two or more decimal points;
--     if there are two or more sign designations;
--     if there exists a character other than '0'..'9' or '.'
--        or '+', '-', ' ' for the sign
--     if the order of the characters is anything other than
--        sign designation followed by the number
function To_SQL_Decimal (Right : SQL_Char_Not_Null) return SQL_Decimal;
function To_SQL_Decimal (Right : SQL_Char) return SQL_Decimal;
-- pragma INLINE(To_SQL_Decimal);

-- The following functions convert from Decimal to Integer
function To_SQL_Int_Not_Null (Right : SQL_Decimal_Not_Null)
    return SQL_Int_Not_Null;
function To_SQL_Int_Not_Null (Right : SQL_Decimal)
    return SQL_Int_Not_Null;
-- pragma INLINE(To_SQL_Int_Not_Null);
function To_SQL_Int (Right : SQL_Decimal) return SQL_Int;
-- pragma INLINE(To_SQL_Int);

-- The following functions convert from Decimal to Float:
function To_SQL_Double_Precision_Not_Null (Right : SQL_Decimal_Not_Null)
    return SQL_Double_Precision_Not_Null;
```

```
-- pragma INLINE (Is_In_Base);

function Is_Null(Value : SQL_Decimal) return boolean;
-- pragma INLINE (Is_Null);
function Not_Null(Value : SQL_Decimal) return boolean;
-- pragma INLINE (Not_Null);

-- The following unary arithmetic operators are provided:
function "+"   (Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "+"   (Right : SQL_Decimal) return SQL_Decimal;
function "-"   (Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "-"   (Right : SQL_Decimal) return SQL_Decimal;
function "abs" (Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "abs" (Right : SQL_Decimal) return SQL_Decimal;
-- pragma INLINE("abs");

-- The following binary arithmetic operators are provided:

-- The "+" and "-" functions return a result with a scale of
--     max(Left.scale, Right.scale)
-- If the operation produces a result that is too large to
--     be represented in an object that has this scale, a
--     Constraint_Error will be raised
function "+" (Left, Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "+" (Left, Right : SQL_Decimal) return SQL_Decimal;
-- pragma INLINE("+");
function "-" (Left, Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "-" (Left, Right : SQL_Decimal) return SQL_Decimal;
-- pragma INLINE("-");
-- The "*" function returns a result with the scale
--    Left.scale + Right.scale
-- If the result is too large to be represented in an object
--    that has this scale, Constraint_Error will be raised
function "*" (Left, Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "*" (Left, Right : SQL_Decimal) return SQL_Decimal;
-- The "/" function returns a result with as much scale as
--    possible, given the nature of the result
-- If the result is too large to be represented in the
--    the underlying hardware or in an object with no scale,
--    or if an attempt is made to divide by zero, the
--    Constraint_Error exception will be raised
function "/" (Left, Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "/" (Left, Right : SQL_Decimal) return SQL_Decimal;

-- The following mixed mode operators are provided:
function "*"   (Left : SQL_Decimal_Not_Null; Right : SQL_Int_Not_Null)
    return SQL_Decimal_Not_Null;
function "*"   (Left : SQL_Decimal; Right : SQL_Int_Not_Null)
    return SQL_Decimal;
function "*"   (Left : SQL_Decimal; Right : SQL_Int)
    return SQL_Decimal;
function "*"   (Left : SQL_Int_Not_Null; Right : SQL_Decimal_Not_Null)
    return SQL_Decimal_Not_Null;
function "*"   (Left : SQL_Int_Not_Null; Right : SQL_Decimal)
    return SQL_Decimal;
function "*"   (Left : SQL_Int; Right : SQL_Decimal)
```

```
--    values
function Zero return SQL_Decimal_Not_Null;
function Zero return SQL_Decimal;
-- pragma INLINE(Zero);
function One return SQL_Decimal_Not_Null;
function One return SQL_Decimal;
-- pragma INLINE(One);

-- The following Assignment procedure is provided for the
--    SQL_Decimal_Not_Null type:
-- The following Assignment procedure raises Constraint_Error
--    if the value of Right does not fall within the range
--    of lower..upper
procedure Assign_With_Check (Left : in out SQL_Decimal_Not_Null;
                             Right : SQL_Decimal_Not_Null;
                             Lower, Upper : SQL_Decimal_Not_Null2);

-- The following Assign_with_check procedure will be used
--    in the generic Assign produced in SQL_Decimal_Ops
-- this procedure raises the Constraint_Error exception if
--    the "Right" input parameter falls outside the range
--    defined by Lower..Upper
procedure Assign_With_Check
                       (Left  : in out SQL_Decimal;
                        Right : SQL_Decimal;
                        Lower, Upper : SQL_Decimal_Not_Null2);
-- pragma INLINE(Assign_with_check);

-- The following comparison operators are provided:

function "="  (Left, Right : SQL_Decimal_Not_Null) return boolean;
function "="  (Left, Right : SQL_Decimal) return boolean;
-- pragma INLINE("=");
function Equals (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE(Equals);
function Not_Equals (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE(Not_Equals);
function "<"  (Left, Right : SQL_Decimal_Not_Null) return boolean;
function "<"  (Left, Right : SQL_Decimal) return boolean;
function "<"  (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE("<");
function ">"  (Left, Right : SQL_Decimal_Not_Null) return boolean;
function ">"  (Left, Right : SQL_Decimal) return boolean;
function ">"  (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE(">");
function "<=" (Left, Right : SQL_Decimal_Not_Null) return boolean;
function "<=" (Left, Right : SQL_Decimal) return boolean;
function "<=" (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE("<=");
function ">=" (Left, Right : SQL_Decimal_Not_Null) return boolean;
function ">=" (Left, Right : SQL_Decimal) return boolean;
function ">=" (Left, Right : SQL_Decimal) return Boolean_With_Unknown;
-- pragma INLINE(">=");

-- the following functions are membership tests
--     the value of the object is tested to see if
--     if it falls within the range of Lower..Upper
function Is_In_Base (Right : SQL_Decimal_Not_Null;
                     Lower, Upper : SQL_Decimal_Not_Null2)
    return boolean;
function Is_In_Base (Right : SQL_Decimal;
                     Lower, Upper : SQL_Decimal_Not_Null2)
    return boolean;
```

```
            SQL_Double_Precision_Not_Null(Without_Null_Type'LAST));
    end assign;

end SQL_Double_Precision_Ops;

end SQL_Double_Precision_Pkg;
```

C.18 SQL_Decimal_Pkg Specification

```
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Int_Pkg; use SQL_Int_Pkg;
with SQL_Char_Pkg; use SQL_Char_Pkg;
with SQL_Double_Precision_Pkg; use SQL_Double_Precision_Pkg;
package SQL_Decimal_Pkg is

    -- MAX_DIGITS is implementation defined
    -- It represents the maximum number of digits that can be
    --     stored in the underlying hardware's representation of
    --     a BCD number
    MAX_DIGITS : constant integer := 31;

    subtype decimal_digits is natural range 0..MAX_DIGITS;

    type SQL_Decimal_Not_Null(scale : decimal_digits := 0) is limited private;
    type SQL_Decimal(scale : decimal_digits) is limited private;

    subtype Numeric_Character is Character range '0'..'9';
    type Numeric_String is array (decimal_digits range <>) of Numeric_Character;
    type Sign_Character is ('+', '-');

    -- the following type is used for purposes of creating generic
    --     assign and is_in functions....DO NOT USE THIS TYPE to
    --     create the abstract domains.....
    type SQL_Decimal_Not_Null2(scale : decimal_digits := 0) is limited private;

    function To_SQL_Decimal_Not_Null (Value : SQL_Decimal_Not_Null2)
        return SQL_Decimal_Not_Null;
    function To_SQL_Decimal (Value : SQL_Decimal_Not_Null2)
        return SQL_Decimal;
    function To_SQL_Decimal_Not_Null2 (Value : SQL_Decimal_Not_Null)
        return SQL_Decimal_Not_Null2;
    function To_SQL_Decimal_Not_Null2 (Value : SQL_Decimal)
        return SQL_Decimal_Not_Null2;
    -- pragma INLINE(To_SQL_Decimal_Not_Null2);

    -- this function returns a null value of the SQL_Decimal type
    function Null_SQL_Decimal return SQL_Decimal;
    -- pragma INLINE(Null_SQL_Decimal);

    -- The following functions shift the value of the object
    --    without changing the scale.  Effectively, the operation
    --    mutiplies the value in the object by 10**Scale.
    -- The following functions raise Constraint_Error if the left
    --    shift causes a loss of significant digits
    function Shift (Value : SQL_Decimal_Not_Null;
                    Scale : integer) return SQL_Decimal_Not_Null;
    function Shift (Value : SQL_Decimal;
                    Scale : integer) return SQL_Decimal;
    -- pragma INLINE(Shift);

    -- The following functions return objects with the appropriate
    function Is_Null(Value : SQL_Int) return Boolean;
    -- pragma INLINE (Is_Null);
    function Not_Null(Value : SQL_Int) return Boolean;
    -- pragma INLINE (Not_Null);

    -- These functions of class type -> boolean
    -- equate UNKNOWN with FALSE. That is, they return TRUE
    -- only when the function returns TRUE. UNKNOWN and FALSE
    -- are mapped to FALSE.
    function "=" (Left, Right : SQL_Int) return Boolean;
    -- pragma INLINE ("=");
    function "<" (Left, Right : SQL_Int) return Boolean;
    -- pragma INLINE ("<");
    function ">" (Left, Right : SQL_Int) return Boolean;
    -- pragma INLINE (">");
    function "<=" (Left, Right : SQL_Int) return Boolean;
    -- pragma INLINE ("<=");
    function ">=" (Left, Right : SQL_Int) return Boolean;
    -- pragma INLINE (">=");


    -- this generic is instantiated once for every abstract
    --   domain based on the SQL type Int.
    -- the three subprogram formal parameters are meant to
    --   default to the programs declared above.
    -- that is, the package should be instantiated in the
    --   scope of a use clause for SQL_Int_Pkg.
    -- the two actual types together form the abstract
    --   domain.
    -- the purpose of the generic is to create functions
    --   which convert between the two actual types and a
    --   procedure which implements a range constrained
    --   assignment for the null-bearing type.
    -- the bodies of these subprograms are calls to
    --   subprograms declared above and passed as defaults to
    --   the generic.
generic
    type With_Null_type is limited private;
    type Without_null_type is range <>;
    with function With_Null_Base(Value : SQL_Int_Not_Null)
        return With_Null_Type is <>;
    with function Without_Null_Base(Value : With_Null_Type)
        return SQL_Int_Not_Null is <>;
    with procedure Assign_with_check (
        Left : in out With_Null_Type; Right : With_Null_Type;
        First, Last : SQL_Int_Not_Null) is <>;
package SQL_Int_Ops is
    function With_Null (Value : Without_Null_type)
        return With_Null_type;
    -- pragma INLINE (With_Null);
    function Without_Null (Value : With_Null_Type)
        return Without_Null_type;
    -- pragma INLINE (Without_Null);
    procedure assign (Left : in out With_null_Type;
                      Right : in With_null_type);
    -- pragma INLINE (assign);
end SQL_Int_Ops;

private

    type SQL_Int is record
        Is_Null: Boolean := true;
        Value: SQL_Int_Not_Null;
          end record;

      end SQL_Int_Pkg;
```

```
-- raises constraint_error if not
--    (First <= Right <= Last)
procedure Assign_with_check (
    Left : in out SQL_Int; Right : SQL_Int;
    First, Last : SQL_Int_Not_Null);
-- pragma INLINE (Assign_with_check);

-- the following functions implement three valued
--    arithmetic
-- if either input to any of these functions is null
--    the function returns the null value; otherwise
--    they perform the indicated operation
-- these functions raise no exceptions
function "+"(Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("+");
function "-"(Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("-");
function "abs"(Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("abs");
function "+"(Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("+");
function "*"(Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("*");
function "-"(Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("-");
function '/"(Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("/");
function "mod" (Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("mod");
function "rem" (Left, Right : SQL_Int) return SQL_Int;
-- pragma INLINE ("rem");
function "**" (Left : SQL_Int; Right: Integer) return SQL_Int;
-- pragma INLINE ("**");

-- simulation of 'IMAGE and 'VALUE that
-- return/take SQL_Char[_Not_Null] instead of string
function IMAGE (Left : SQL_Int_Not_Null) return SQL_Char_Not_Null;
function IMAGE (Left : SQL_Int) return SQL_Char;
function VALUE (Left : SQL_Char_Not_NULL) return SQL_Int_Not_Null;
function VALUE (Left : SQL_Char) return SQL_Int;

    -- Logical Operations --
    -- type X type => Boolean_with_unknown --
-- these functions implement three valued logic
-- if either input is the null value, the functions
--    return the truth value UNKNOWN; otherwise they
--    perform the indicated comparison.
-- these functions raise no exceptions
function Equals (Left, Right : SQL_Int) return Boolean_with_Unknown;
-- pragma INLINE (Equals);
function Not_Equals (Left, Right : SQL_Int)
                        return Boolean_with_Unknown;
-- pragma INLINE (Not_Equals);
function "<" (Left, Right : SQL_Int) return Boolean_with_Unknown;
-- pragma INLINE ("<");
function ">" (Left, Right : SQL_Int) return Boolean_with_Unknown;
-- pragma INLINE (">");
function "<=" (Left, Right : SQL_Int) return Boolean_with_Unknown;
-- pragma INLINE ("<=");
function ">=" (Left, Right : SQL_Int) return Boolean_with_Unknown;
-- pragma INLINE (">=");

    -- type => boolean --
```

```
function "xor" (Left, Right : Boolean_with_Unknown)
                return Boolean_with_Unknown is
begin
    return (Left and not Right) or (not Left and Right);
end;

--- three-val => bool or exception ---
function To_Boolean (Left : Boolean_with_Unknown) return Boolean is
begin
if Left = Unknown then raise null_value_error;
else return (Left = True);
end if;
end;

--- three-val => bool ---
function Is_True (Left : Boolean_with_Unknown) return Boolean is
begin
    return (Left = True);
end;
function Is_False (Left : Boolean_with_Unknown) return Boolean is
begin
    return (Left = False);
end;
function Is_Unknown (Left : Boolean_with_Unknown) return Boolean is
begin
    return (Left = Unknown);
end;

end SQL_Boolean_Pkg;
```

C.10 SQL_Int_Pkg Specification

```
with SQL_Standard;
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Char_Pkg; use SQL_Char_Pkg;
package SQL_Int_Pkg
    is

    type SQL_Int_not_null is new SQL_Standard.Int;

        ---- Possibly Null Integer ----
    type SQL_Int is limited private;

    function Null_SQL_Int return SQL_Int;
    -- pragma INLINE (Null_SQL_Int);

    -- this pair of functions convert between the
    --   null-bearing and non-null-bearing types.
    function Without_Null_Base(Value : SQL_Int)
      return SQL_Int_Not_Null;
    -- pragma INLINE (Without_Null_Base);
    -- With_Null_Base raises Null_Value_Error if the input
    --   value is null
    function With_Null_Base(Value : SQL_Int_Not_Null)
      return SQL_Int;
    -- pragma INLINE (With_Null_Base);

    -- this procedure implements range checking
    -- note: it is not meant to be used directly
    --   by application programmers
    -- see the generic package SQL_Int_Ops
    -- pragma INLINE ("or");
    function "xor" (Left, Right : Boolean_with_Unknown)
                   return Boolean_with_Unknown;
    -- pragma INLINE ("xor");

    --- three-val => bool or exception ---
    function To_Boolean (Left : Boolean_with_Unknown) return Boolean;
    -- pragma INLINE (To_Boolean);

    --- three-val => bool ---
    function Is_True (Left : Boolean_with_Unknown) return Boolean;
    -- pragma INLINE (Is_True);
    function Is_False (Left : Boolean_with_Unknown) return Boolean;
    -- pragma INLINE (Is_False);
    function Is_Unknown (Left : Boolean_with_Unknown) return Boolean;
    -- pragma INLINE (Is_Unknown);

end SQL_Boolean_Pkg;
```

C.9 SQL_Boolean_Pkg Body

```
With SQL_Exceptions;
package body SQL_Boolean_Pkg is

    Null_Value_Error : exception renames SQL_Exceptions.Null_Value_Error;

    function "not"  (Left : Boolean_with_Unknown)
                   return Boolean_with_Unknown is
    begin
    case Left is
        when true => return false;
        when false => return true;
        when unknown => return unknown;
    end case;
    end;

    function "and" (Left, Right : Boolean_with_Unknown)
                   return Boolean_with_Unknown is
    begin
    if (Left = False) or else (Right = False) then
        return False;
    elsif (Left = Unknown) or else (Right = Unknown) then
        return Unknown;
    else
        return True;
    end if;
    end;

    function "or" (Left, Right : Boolean_with_Unknown)
                   return Boolean_with_Unknown is
    begin
    if (Left = True) or else (Right = True) then
        return True;
    elsif (Left = Unknown) or else (Right = Unknown) then
        return Unknown;
    else
        return False;
    end if;
    end;
```

```
        -- pragma INLINE(Is_Day_Time);
        function Not_Year_Month(Value : SQL_Interval) return Boolean;
        -- pragma INLINE(Not_Year_Month);
        function Not_Day_Time(Value : SQL_Interval) return Boolean;
        -- pragma INLINE (Not_Day_Time);

        -- the procedure Current returns the current system Datetime, using
        --    the precision of the input variable
        procedure Current (Value : in out SQL_Date);
        -- pragma INLINE(Current);
        -- the procedure Extend returns the value of the Right input object with
        --    the datetime qualifier of the Left object, if a valid datetime
        --    value is generated by the extension process
        procedure Extend (Value : in out SQL_Date);
        -- pragma INLINE(Extend);

        -- this generic is instantiated once for every abstract
        --    SQL_Date domain, and once for every abstract SQL_Interval
        --    domain, based on the type SQL_Date_Not_Null.
        -- the two subprogram formal parameters are meant to
        --    default to the programs declared above.
        -- that is, the package should be instantiated in the
        --    scope of a use clause for SQL_Date_Pkg.
        -- the two actual types together form the abstract
        --    domain.
        -- the purpose of the generic is to create functions
        --    which convert between the two actual types
        -- the bodies of these subprograms are calls to
        --    subprograms declared above and passed as defaults to
        --    the generic.
    generic
        type With_Null_Type is limited private;
        type Without_Null_Type is array (positive range <>)
          of SQL_Standard.Character_type;
        with procedure Parse_and_Assign_Base
            (Left : in out With_Null_Type; Right : SQL_Date_Not_Null) is <>;
        with function Without_Null_Base(Value : With_Null_Type)
            return SQL_Date_Not_Null is <>;
    package SQL_Date_Ops is
        procedure Parse_and_Assign (Left : With_Null_Type;
                                    Right : Without_Null_Type);
        -- pragma INLINE (Parse_and_Assign);
        function Without_Null (Value : With_Null_Type)
            return Without_Null_type;
        -- pragma INLINE (Without_Null);
    end SQL_Date_Ops;

    generic
        type With_Null_Date_Type is limited private;
        type With_Null_Interval_Type is limited private;
        with function Plus (Left : With_Null_Date_Type; Right : SQL_Interval)
                return With_Null_Date_Type is <>;
        with function Plus (Left : SQL_Interval; Right : With_Null_Date_Type)
                return With_Null_Date_Type is <>;
        with function Minus (Left : With_Null_Date_Type; Right : SQL_Interval)
                return With_Null_Date_Type is <>;
        with function Minus (Left, Right : With_Null_Date_Type)
                return SQL_Interval is <>;
    package SQL_Date_Interval_Ops is
        function "+" (Left : With_Null_Date_Type; Right : With_Null_Interval_Type)
            return With_Null_Date_Type;
        function "+" (Left : With_Null_Interval_Type; Right : With_Null_Date_Type)
            return With_Null_Date_Type;
    function "-" (Left : With_Null_Date_Type; Right : With_Null_Interval_Type)
        return With_Null_Date_Type;
    function "-" (Left, Right : With_Null_Date_Type)
        return With_Null_Interval_Type;
end SQL_Date_Interval_Ops;

private

    type SQL_year_number     is range 1600..9999;
    type SQL_month_number    is range 1..12;
    type SQL_day_number      is range 1..31;
    type SQL_hour_number     is range 0..23;
    type SQL_minute_number   is range 0..59;
    type SQL_second_number   is range 0..59;
    type SQL_fraction_number is range 0..(2**31)-1;
    type SQL_interval_number is range -(2**31)..(2**31)-1;

    type SQL_Date(From       : SQL_Datetime_Field;
                  To         : SQL_Datetime_Field;
                  Fractional : precision)
    is record
        Is_Null  : Boolean := true;
        year     : SQL_year_number;
        month    : SQL_month_number;
        day      : SQL_day_number;
        hour     : SQL_hour_number;
        minute   : SQL_minute_number;
        second   : SQL_second_number;
        fraction : SQL_fraction_number;
    end record;

    type SQL_Interval(From       : SQL_Datetime_Field;
                      Leading    : precision;
                      To         : SQL_Datetime_Field;
                      Fractional : precision)
    is record
        Is_Null       : boolean := True;
        Is_Year_Month : boolean := True;
        years         : SQL_interval_number;
        months        : SQL_interval_number;
        days          : SQL_interval_number;
        minutes       : SQL_interval_number;
        seconds       : SQL_interval_number;
        fraction      : SQL_interval_number;
    end record;

end SQL_Date_Pkg;
```

C.31 INGRES_Date_Pkg Specification

```
with SQL_Standard;
with SQL_System; use SQL_System;
with Calendar; use Calendar;
with SQL_Boolean_Pkg; use SQL_Boolean_Pkg;
with SQL_Char_Pkg; use SQL_Char_Pkg;
package INGRES_Date_Pkg
    is

    type INGRES_Date_Not_Null is new SQL_Char_Not_Null;
        ---- Possibly Null Datetime ----
```

Package SQL_Base_Types_Pkg

```
with SQL_Char_Pkg, SQL_Int_Pkg, SQL_Smallint_Pkg, SQL_Real_Pkg,
     SQL_Double_Precision_Pkg, SQL_Decimal_Pkg, SQL_Standard;
package SQL_Base_Types_Pkg is

    package Character_Set renames SQL_Standard.Character_Set;

    type SQL_Int_Not_Null is new SQL_Int_Pkg.SQL_Int_Not_Null;
    type SQL_Int_Type is new SQL_Int_Pkg.SQL_Int;
    package SQL_Int_Ops is new SQL_Int_Pkg.SQL_Int_Ops(
            SQL_Int_Type, SQL_Int_Not_Null);
    subtype SQL_Int_Subtype is SQL_Int_Pkg.SQL_Int;
    subtype SQL_Int_Not_Null_Subtype is SQL_Int_Pkg.SQL_Int_Not_Null;

   type SQL_Smallint_Not_Null is new SQL_Smallint_Pkg.SQL_Smallint_Not_Null;
    type SQL_Smallint_Type is new SQL_Smallint_Pkg.SQL_Smallint;
    package SQL_Smallint_Ops is new SQL_Smallint_Pkg.SQL_Smallint_Ops(
            SQL_Smallint_Type, SQL_Smallint_Not_Null);
    subtype SQL_Smallint_Subtype is SQL_Smallint_Pkg.SQL_Smallint;
    subtype SQL_Smallint_Not_Null_Subtype is
            SQL_Smallint_Pkg.SQL_Smallint_Not_Null;

    type SQL_Real_Not_Null is new SQL_Real_Pkg.SQL_Real_Not_Null;
    type SQL_Real_Type is new SQL_Real_Pkg.SQL_Real;
    package SQL_Real_Ops is new SQL_Real_Pkg.SQL_Real_Ops(
            SQL_Real_Type, SQL_Real_Not_Null);
    subtype SQL_Real_Subtype is SQL_Real_Pkg.SQL_Real;
    subtype SQL_Real_Not_Null_Subtype is SQL_Real_Pkg.SQL_Real_Not_Null;

    type SQL_Double_Precision_Not_Null is
        new SQL_Double_Precision_Pkg.SQL_Double_Precision_Not_Null;
    type SQL_Double_Precision_Type is
        new SQL_Double_Precision_Pkg.SQL_Double_Precision;
    package SQL_Double_Precision_Ops is
        new SQL_Double_Precision_Pkg.SQL_Double_Precision_Ops(
            SQL_Double_Precision_Type,
                SQL_Double_Precision_Not_Null);
    subtype SQL_Double_Precision_Subtype is
            SQL_Double_Precision_Pkg.SQL_Double_Precision;
    subtype SQL_Double_Precision_Not_Null_Subtype is
            SQL_Double_Precision_Pkg.SQL_Double_Precision_Not_Null;

    type SQL_Char_Not_Null is new SQL_Char_Pkg.SQL_Char_Not_Null;
    type SQL_Char_Type is new SQL_Char_Pkg.SQL_Char;
    package SQL_Char_Ops is new SQL_Char_Pkg.SQL_Char_Ops(
            SQL_Char_Type, SQL_Char_Not_Null);
    subtype SQL_Char_Subtype is SQL_Char_Pkg.SQL_Char;
    subtype SQL_Char_Not_Null_Subtype is SQL_Char_Pkg.SQL_Char_Not_Null;

    type SQL_Decimal_Not_Null is new SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
    type SQL_Decimal_Type is new SQL_Decimal_Pkg.SQL_Decimal;
    package SQL_Decimal_Ops is new SQL_Decimal_Pkg.SQL_Decimal_Ops(
            SQL_Decimal_Type, SQL_Decimal_Not_Null);
    subtype SQL_Decimal_Subtype is SQL_Decimal_Pkg.SQL_Decimal;
    subtype SQL_Decimal_Not_Null_Subtype is
            SQL_Decimal_Pkg.SQL_Decimal_Not_Null;

end SQL_Base_Types_Pkg;
```

Package SQL_Standard_Dynamic

```
with SQL_Standard, SQL_Decimal_Pkg;
use SQL_Standard, SQL_Decimal_Pkg;
package SQL_Standard_Dynamic is

type Extended_Cursor_Type is implementation defined;
type Extended_Statement_Type is implementation defined;
type SQL_Dynamic_Datatypes_Base is range implementation defined;

Maybe_Null_Indic : constant  Indicator_Type := 1;
       -- value of SQLNULLABLE if nulls allowed
subtype Null_Indication is Indicator_Type range
     Indicator_Type'First .. -1;
       -- value of indicator if value is null

    -- types to describe column names
SQL_Column_Name_Length : constant := 18; -- set in SQL2 standard
subtype SQL_Column_Name_Length_Type is
        positive range 1..SQL_Column_Name_Length;
subtype SQLNAME_Type is Char(SQL_Column_Name_Length_Type);

-- These constants capture the encoding of SQL Types as integers
-- as given by SQL2.
Not_Specified :             constant SQL_Dynamic_Datatypes_Base := 0;
Dynamic_Char :              constant SQL_Dynamic_Datatypes_Base := 1;
Dynamic_Numeric :           constant SQL_Dynamic_Datatypes_Base := 2;
Dynamic_Decimal :           constant SQL_Dynamic_Datatypes_Base := 3;
Dynamic_Int :               constant SQL_Dynamic_Datatypes_Base := 4;
Dynamic_Smallint :          constant SQL_Dynamic_Datatypes_Base := 5;
Dynamic_Float:              constant SQL_Dynamic_Datatypes_Base := 6;
Dynamic_Real :              constant SQL_Dynamic_Datatypes_Base := 7;
Dynamic_Double_Precision : constant SQL_Dynamic_Datatypes_Base := 8;

subtype SQL_Dynamic_Datatypes is SQL_Dynamic_Datatypes_Base
    range Not_Specified .. Dynamic_Double_Precision;

   -- access types for components of SQL_Dynamic_Parameter
type Char_Access is access Char;
type Decimal_Access is access SQL_Decimal_Not_Null;
type Int_Access is access Int;
type Smallint_Access is access Smallint;
type Real_Access is access Real;
type Double_Precision_Access is access Double_Precision;
type SQL_Dynamic_Parameter (SQLTYPE :SQL_Dynamic_Datatypes:=Not_Specified)
   is record
        case SQLType is
             when Not_Specified =>
                 null;
             when Dynamic_Char =>
                 Char_Value : Char_Access;
             when Dynamic_Decimal | Dynamic_Numeric =>
                 Decimal_Value : Decimal_Access;
             when Dynamic_Int =>
                 Int_Value : Int_Access;
             when Dynamic_Smallint =>
                 Smallint_Value : Smallint_Access;
             when Dynamic_Real =>
                 Real_Value : Real_Access;
             when Dynamic_Double_Precision  | Dynamic_Float =>
                 Double_Precision_Value : Double_Precision_Access;
        end case;
    end record;

type SQLVAR_Component_Type is record
```

```
     SQLDATA : SQL_Dynamic_Parameter;
     SQLNULLABLE : Indicator_Type;
     SQLIND : Indicator_Type;
     SQLNAMEL : SQL_Column_Name_Length_Type;
     SQLNAME : SQLNAME_Type;
end record;

type SQLVAR_Type is
     array (Int range <>) of SQLVar_Component_Type;

type SQLDA (SQLN : Int) is record
     SQLD : Int;
     SQLVAR : SQLVAR_Type (1 .. SQLN);
end record;

end SQL_Standard_Dynamic;
```

Package SQL_Dynamic_Pkg

```
with SQL_Base_Types_Pkg, SQL_Standard_Dynamic;
use SQL_Base_Types_Pkg;
package SQL_Dynamic_Pkg is

-- These next definitions deal with names of columns
subtype SQL_Column_Name_Length_Type is
        positive range 1..SQL_Standard_Dynamic.SQL_Column_Name_Length;
subtype SQLNAME_Type is SQL_Char_Not_Null(SQL_Column_Name_Length_Type);

-- The discriminant is now an enumeration type
type SQL_Dynamic_Datatypes is
     (Not_Specified,
      Dynamic_Char, Dynamic_Decimal,
      Dynamic_Int, Dynamic_Smallint,
      Dynamic_Real, Dynamic_Double_Precision);

-- access types access null bearing types in Base_Type_Pkg
type Char_Access is access SQL_Char_Type;
type Decimal_Access is access SQL_Decimal_Type;
type Int_Access is access SQL_Int_Type;
type Smallint_Access is access SQL_Smallint_Type;
type Real_Access is access SQL_Real_Type;
type Double_Precision_Access is access SQL_Double_Precision_Type;


type SQL_Dynamic_Parameter (SQLTYPE :SQL_Dynamic_Datatypes := Not_Specified
   is record
        case SQLType is
             when Not_Specified =>
                  null;
             when Dynamic_Char  =>
                  Char_Value : Char_Access;
             when Dynamic_Decimal =>
                  Decimal_Value : Decimal_Access;
             when Dynamic_Int =>
                  Int_Value : Int_Access;
             when Dynamic_Smallint =>
                  Smallint_Value : Smallint_Access;
             when Dynamic_Real =>
                  Real_Value : Real_Access;
             when Dynamic_Double_Precision =>
                  Double_Precision_Value : Double_Precision_Access;
        end case;
   end record;

type SQLVAR_Component_Type is record
     SQLDATA : SQL_Dynamic_Parameter;
     SQLNAMEL : SQL_Column_Name_Length_Type;
     SQLNAME : SQLNAME_Type;
end record;

type SQLVAR_Type is
     array (SQL_Int_Not_Null range <>) of SQLVar_Component_Type;

type SQLDA (SQLN : SQL_Int_Not_Null) is record
     SQLD : SQL_Int_Not_Null;
     SQLVAR : SQLVAR_Type (1 .. SQLN);
end record;
end SQL_Dynamic_Pkg;
```

Appendix C. Abstract Interface Domain Primitive Types Code

The following listing provides the abstract interface code for the Domain Primitive Types that was developed and discussed in Chapter IV. For ease of understanding, the information is organized in a form that is not compilable. To clarify how each attribute contributes to this package, each attribute's information is consolidated in one area which is separated from other attribute information areas by a blank line. To implement these packages all generic "SQL_*_Pkg.SQL_*_OPS(...)" packages must be moved to the end of each entity package since they are later declarative items. Each Primitive Type Package begins on a new page.

```
     .
     .
     .

*Package* The_Area_primitive_domain_types
     .
     .

with SQL_Int_Pkg,
     SQL_Char_Pkg;
package The_Area_primitive_domain_types is
     .
    type AREA_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type AREA_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package AREA_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (AREA_ID_Type, AREA_ID_Not_Null );

    type DOMAINNN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype DOMAIN_Not_Null is
                DOMAINNN_Base
                (1..256);
    type DOMAIN_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype DOMAIN_Type is
                DOMAIN_Base
                (DOMAIN_Not_Null'Length);
    package DOMAIN_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (DOMAIN_Base, DOMAIN_Not_Null );

    type SEANN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype SEA_Not_Null is
                SEANN_Base
                (1..256);
    type SEA_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype SEA_Type is
                SEA_Base
                (SEA_Not_Null'Length);
    package SEA_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (SEA_Base, SEA_Not_Null );
    type PHASENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype PHASE_Not_Null is
                PHASENN_Base
                (1..256);
    type PHASE_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype PHASE_Type is
                PHASE_Base
                (PHASE_Not_Null'Length);

    package PHASE_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (PHASE_Base, PHASE_Not_Null );

end The_Area_primitive_domain_types;
```

Package General_Software_Characteristic_primitive_domain_types

```
with SQL_Int_Pkg,
     SQL_Char_Pkg,
     SQL_Enumeration_Pkg,
package General_Software_Characteristic_primitive_domain_types is

    type GSC_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type GSC_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package GSC_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (GSC_ID_Type, GSC_ID_Not_Null );

    type CHARACT_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype CHARACT_NAME_Not_Null is
                CHARACT_NAMENN_Base
                (1..256);
    type CHARACT_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype CHARACT_NAME_Type is
                CHARACT_NAME_Base
                (CHARACT_NAME_Not_Null'Length);
    package CHARACT_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (CHARACT_NAME_Base, CHARACT_NAME_Not_Null );

    type formu_?NN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype formu_?_Not_Null is
                formu_?NN_Base
                (1..256);
```

```
type formu_?_Base is new
   SQL_Char_Pkg.SQL_Char;
   subtype formu_?_Type is
         formu_?_Base
         (formu_?_Not_Null'Length);
package formu_?_Ops is new
   SQL_Char_Pkg.SQL_Char_Ops
   (formu_?_Base, formu_?_Not_Null );

type evalu_?NN_Base is new
   SQL_Char_Pkg.SQL_Char_Not_Null;
   subtype evalu_?_Not_Null is
         evalu_?NN_Base
         (1..256);
type evalu_?_Base is new
   SQL_Char_Pkg.SQL_Char;
   subtype evalu_?_Type is
         evalu_?_Base
         (evalu_?_Not_Null'Length);
package evalu_?_Ops is new
   SQL_Char_Pkg.SQL_Char_Ops
   (evalu_?_Base, evalu_?_Not_Null );

type evalu_helpNN_Base is new
   SQL_Char_Pkg.SQL_Char_Not_Null;
   subtype evalu_help_Not_Null is
         evalu_helpNN_Base
         (1..256);
type evalu_help_Base is new
   SQL_Char_Pkg.SQL_Char;
   subtype evalu_help_Type is
         evalu_help_Base
         (evalu_help_Not_Null'Length);
package evalu_help_Ops is new
   SQL_Char_Pkg.SQL_Char_Ops
   (evalu_help_Base, evalu_help_Not_Null ):

type essential_flag_Not_Null is (Is_False, Is_true):
package essential_flag_Pkg is new
   SQL_Enumeration_Pkg
   (essential_flag_Not_Null);
type essential_flag_Type is new
   essential_flag_Pkg.SQL_Enumeration_Pkg;

type evalu_method_Not_Null is (Is_False, Is_true);
package evalu_method_Pkg is new
   SQL_Enumeration_Pkg
   (evalu_method_Not_Null);
type evalu_method_Type is new
   evalu_method_Pkg.SQL_Enumeration_Pkg;
```

```
    type empirical_weight_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type empirical_weight_Type is new
        SQL_Int_Pkg.SQL_Int;
    package empirical_weight_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (empirical_weight_Type, empirical_weight_Not_Null );

end General_Software_Characteristic_primitive_domain_types;
```

Package The_Tool_primitive_domain_types

```
with SQL_Int_Pkg,
     SQL_Char_Pkg,
     SQL_Decimal_Pkg;
package The_Tool_primitive_domain_types is

    type TOOL_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type TOOL_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package TOOL_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (TOOL_ID_Type, TOOL_ID_Not_Null );

    type TOOL_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype TOOL_NAME_Not_Null is
                TOOL_NAMENN_Base
                (1..256);
    type TOOL_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype TOOL_NAME_Type is
                TOOL_NAME_Base
                (TOOL_NAME_Not_Null'Length);
    package TOOL_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (TOOL_NAME_Base, TOOL_NAME_Not_Null );

    version_scale: constant decimal_digits:= 2;
    type VERSIONNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype VERSION_Not_Null is
                VERSIONNN_Base
                (scale => version_scale);
    type VERSION_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype VERSION_Type is
                VERSION_Base
                (scale => version_scale);
    package VERSION_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (VERSION_Base,
        VERSIONNN_Base,
        in_scale => version_scale);
    type vendorNN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype vendor_Not_Null is
                vendorNN_Base
                (1..256);
    type vendor_Base is new
        SQL_Char_Pkg.SQL_Char;
        subtype vendor_Type is
                vendor_Base
                (vendor_Not_Null'Length);
    package vendor_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (vendor_Base, vendor_Not_Null );

    cost_scale: constant decimal_digits:= 2;
    type costNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype cost_Not_Null is
                costNN_Base
                (scale => cost_scale);
    type cost_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype cost_Type is
                cost_Base
                (scale => cost_scale);
    package cost_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (cost_Base,
        costNN_Base,
        in_scale => cost_scale);

end The_Tool_primitive_domain_types;
```

Package The_Evaluator_primitive_domain_types

```
with SQL_Int_Pkg,
     SQL_Char_Pkg,
     SQL_Date_Pkg;
package The_Evaluator_primitive_domain_types is

    type EVAL_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type EVAL_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package EVAL_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (EVAL_ID_Type, EVAL_ID_Not_Null );

    type FIRST_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype FIRST_NAME_Not_Null is
                FIRST_NAMENN_Base
                (1..256);
    type FIRST_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype FIRST_NAME_Type is
                FIRST_NAME_Base
                (FIRST_NAME_Not_Null'Length);
    package FIRST_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (FIRST_NAME_Base, FIRST_NAME_Not_Null );

    type LAST_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype LAST_NAME_Not_Null is
                LAST_NAMENN_Base
                (1..256);
    type LAST_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype LAST_NAME_Type is
                LAST_NAME_Base
                (LAST_NAME_Not_Null'Length);
    package LAST_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (LAST_NAME_Base, LAST_NAME_Not_Null );
    type dateNN_Base is new
        SQL_Date_Pkg.SQL_Date_Not_Null;
        subtype date_Not_Null is
                dateNN_Base (1..10);
    type date_Base is new
        SQL_Date_Pkg.SQL_Date
        (From =>year, To=>Day, Fractional =>0);
    package date_Ops is new
        SQL_Date_Pkg.SQL_Date_Ops
        (date_Type, dateNNBase);

    type typeNN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype type_Not_Null is
                typeNN_Base
                (1..256);
    type type_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype type_Type is
                type_Base
                (type_Not_Null'Length);
    package type_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (type_Base, type_Not_Null );

end The_Evaluator_primitive_domain_types;
```

Package The_Quality_primitive_domain_types

```
with SQL_Int_Pkg,
     SQL_Char_Pkg;
package The_Quality_primitive_domain_types is

    type QUAL_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type QUAL_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package QUAL_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (QUAL_ID_Type, QUAL_ID_Not_Null );

    type QUALITY_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype QUALITY_NAME_Not_Null is
                QUALITY_NAMENN_Base
                (1..256);
    type QUALITY_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype QUALITY_NAME_Type is
                QUALITY_NAME_Base
                (QUALITY_NAME_Not_Null'Length);
    package QUALITY_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (QUALITY_NAME_Base, QUALITY_NAME_Not_Null );

    type QUALITY_VALUENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype QUALITY_VALUE_Not_Null is
                QUALITY_VALUENN_Base
                (1..256);
    type QUALITY_VALUE_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype QUALITY_VALUE_Type is
                QUALITY_VALUE_Base
                (QUALITY_VALUE_Not_Null'Length);
    package QUALITY_VALUE_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (QUALITY_VALUE_Base, QUALITY_VALUE_Not_Null );

end The_Quality_primitive_domain_types;
```

Package The_Specific_Software_Characteristic_primitive_domain_types

```
with SQL_Int_Pkg,
     SQL_Char_Pkg;
package The_Specific_Software_Characteristic_primitive_domain_types is

    type SSC_ID_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type SSC_ID_Type is new
        SQL_Int_Pkg.SQL_Int;
    package SSC_ID_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (SSC_ID_Type, SSC_ID_Not_Null );

    type valueNN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype value_Not_Null is
                valueNN_Base
                (1..256);
    type value_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype value_Type is
                value_Base
                (value_Not_Null'Length);
    package value_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (value_Base, value_Not_Null );

    type tepNN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype tep_Not_Null is
                tepNN_Base
                (1..256);
    type tep_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype tep_Type is
                tep_Base
                (tep_Not_Null'Length);
    package tep_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (tep_Base, tep_Not_Null );

end The_Specific_Software_Characteristic_primitive_domain_types;
```

Package Weight_Set_primitive_domain_types

```
with SQL_Char_Pkg,
     SQL_Enumeration_Pkg;
package Weight_Set_primitive_domain_types is

    type WEIGHT_SET_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype WEIGHT_SET_NAME_Not_Null is
                WEIGHT_SET_NAMENN_Base
                (1..256);
    type WEIGHT_SET_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype WEIGHT_SET_NAME_Type is
                WEIGHT_SET_NAME_Base
                (WEIGHT_SET_NAME_Not_Null'Length);
    package WEIGHT_SET_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (WEIGHT_SET_NAME_Base, WEIGHT_SET_NAME_Not_Null );

    type default_Not_Null is (Is_False, Is_true);
    package default_Pkg is new
        SQL_Enumeration_Pkg
        (default_Not_Null);
    type default_Type is new
        default_Pkg.SQL_Enumeration_Pkg;

end Weight_Set_primitive_domain_types;
```

Package Selection_Set_primitive_domain_types

```
with SQL_Char_Pkg;
package Selection_Set_primitive_domain_types is

    type SET_NAMENN_Base is new
        SQL_Char_Pkg.SQL_Char_Not_Null;
        subtype SET_NAME_Not_Null is
                SET_NAMENN_Base
                (1..256);
    type SET_NAME_Base is new
         SQL_Char_Pkg.SQL_Char;
        subtype SET_NAME_Type is
                SET_NAME_Base
                (SET_NAME_Not_Null'Length);
    package SET_NAME_Ops is new
        SQL_Char_Pkg.SQL_Char_Ops
        (SET_NAME_Base, SET_NAME_Not_Null );

end Selection_Set_primitive_domain_types;
```

Package software_char_score_primitive_domain_types

```
with SQL_Decimal_Pkg;
package software_char_score_primitive_domain_types is

    software_char_function_score_scale: constant decimal_digits:= 2;
    type software_char_function_scoreNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype software_char_function_score_Not_Null is
                software_char_function_scoreNN_Base
                (scale => software_char_function_score_scale);
    type software_char_function_score_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype software_char_function_score_Type is
                software_char_function_score_Base
                (scale => software_char_function_score_scale);
    package software_char_function_score_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (software_char_function_score_Base,
        software_char_function_scoreNN_Base,
        in_scale => software_char_function_score_scale);

    software_char_quality_score_scale: constant decimal_digits:= 2;
    type software_char_quality_scoreNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype software_char_quality_score_Not_Null is
                software_char_quality_scoreNN_Base
                (scale => software_char_quality_score_scale);
    type software_char_quality_score_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype software_char_quality_score_Type is
                software_char_quality_score_Base
                (scale => software_char_quality_score_scale);
    package software_char_quality_score_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (software_char_quality_score_Base,
        software_char_quality_scoreNN_Base,
        in_scale => software_char_quality_score_scale);
end software_char_score_primitive_domain_types;
```

Package tool_score_primitive_domain_types

```
with SQL_Decimal_Pkg;
package tool_score_primitive_domain_types is

    tool_function_score_scale: constant decimal_digits:= 2;
    type tool_function_scoreNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype tool_function_score_Not_Null is
                tool_function_scoreNN_Base
                (scale => tool_function_score_scale);
    type tool_function_score_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype tool_function_score_Type is
                tool_function_score_Base
                (scale => tool_function_score_scale);
    package tool_function_score_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (tool_function_score_Base,
        tool_function_scoreNN_Base,
        in_scale => tool_function_score_scale);

    tool_quality_score_scale: constant decimal_digits:= 2;
    type tool_quality_scoreNN_Base is new
        SQL_Decimal_Pkg.SQL_Decimal_Not_Null;
        subtype tool_quality_score_Not_Null is
                tool_quality_scoreNN_Base
                (scale => tool_quality_score_scale);
    type tool_quality_score_Base is new
        SQL_Decimal_Pkg.SQL_Decimal;
        subtype tool_quality_score_Type is
                tool_quality_score_Base
                (scale => tool_quality_score_scale);
    package tool_quality_score_Ops is new
        SQL_Decimal_Pkg.SQL_Decimal_Ops
        (tool_quality_score_Base,
        tool_quality_scoreNN_Base,
        in_scale => tool_quality_score_scale);

end tool_score_primitive_domain_types;
```

Package software_char_weight_primitive_domain_types

```
with SQL_Int_Pkg;
package software_char_weight_primitive_domain_types is

    type function_weight_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type function_weight_Type is new
        SQL_Int_Pkg.SQL_Int;
    package function_weight_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (function_weight_Type, function_weight_Not_Null );

    type quality_weight_Not_Null is new
        SQL_Int_Pkg.SQL_Int_Not_Null;
    type quality_weight_Type is new
        SQL_Int_Pkg.SQL_Int;
    package quality_weight_Ops is new
        SQL_Int_Pkg.SQL_Int_Ops
        (quality_weight_Type, quality_weight_Not_Null );

end software_char_weight_primitive_domain_types;
```

Appendix D. Abstract Interface Composite Methods Code

The following listing provides the abstract interface composite methods code that was developed and discussed in Chapter IV. The code in this appendix is compilable as long as the SAME packages have already been installed on the users system. The code defines the abstract interface by providing the specifications to that interface, the bodies to these specifications can be implemented using the behavioral description provided in Chapter IV as a guideline. These packages represent the second piece to the logical entity descriptions defined in Chapter IV. Recall that a logical entity represented both the domain primitive type and the methods that operated on that type. The domain primitive types are presented in Appendix C, the methods that operate on those types are presented here.

Simplifying assumptions were:

- A Formulator released SAI structure would not be altered after an evaluator or selector process used that structure. The final system will have to insert abstract interface code that would support this type of system requirement.

- Dynamic SQL statements require a package shell which contained a description of the parameters that are dynamic. This package is required for accessing The_Area, software_char_score, The_Tool, Specific Software Characteristic, and The_Quality combined attributes. This package operates across several domain primitive types to accomplish the STEMdB requirement that the tool selection process allow the user to constrain anywhere from one to several of the attributes located in these domains.

Package Tool_narrowing

```
with    SQL_Base_Types_Pkg,
        SQL_Dynamic_Pkg
        The_Area_primitive_domain_types,
        software_char_score_primitive_domain_types,
        The_Tool_primitive_domain_types,
        Specific Software Characteristic_primitive_domain_types,
        The_Quality_primitive_domain_types,
        The_Evaluator_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        SQL_Dynamic_Pkg
        The_Area_primitive_domain_types,
        software_char_score_primitive_domain_types,
        The_Tool_primitive_domain_types,
        Specific Sofware Characteristic_primitive_domain_types,
        The_Quality_primitive_domain_types,
        The_Evaluator_primitive_domain_types;

Package Tool_narrowing is

        Prepare(STMT: SQL_Char_not_null);
        Allocate(for_SQLDA_name: SQL_Char_not_null;
                Max: SQL_int_not_null );

        Describe_Input(for_SQLDA_in_area: SQL_Char_not_null);

        Describe_output(for_SQLDA_out_area: SQL_Char_not_null);

        Get_Number_parameters(for_given_SQLDA: SQL_Char_not_null;
                              num: out SQL_Int_not_null);

        Get_parameter_type(parameter_number: SQL_Int_not_null;
                           for_given_SQLDA: SQL_Char_not_null;
                           parameter_type: out SQL_Dynamic_Data_types);

        Set_parameter_value(parameter_number: SQL_Int_not_null;
                            for_given_SQLDA: SQL_Char_not_null;
                            SQLDATA: in SQL_Char_type);

        Get_parameter_value(parameter_number: SQL_Int_not_null;
                            from_given_SQLDA: SQL_Char_not_null;
                            SQLDATA: out SQL_Char_type);

        Open_Cursor(for_SQLDA_name: SQL_Char_not_null):

        Fetch(place_in_SQLDA_name: out SQL_Char_not_null:
                value_fetched: out boolean);
        Close_cursor;
end Tool_narrowing;
```

Package Database_Transactions

```
Package Database_Transactions is
--Required initialization routine
        Procedure CreateTables;
--Transactions
        Procedure Commit;
        Proceudure Rollback;
end Database_Transactions;
```

Package The_Area_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Area_primitive_domain_types;


use     SQL_Base_Types_Pkg,
        The_Area_primitive_domain_type;


Package The_Area_Composite_OPS is

        Type area_record_type is record
                AREA_ID: AREA_ID_Not_Null;
                DOMAIN: DOMAIN_Not_Null;
                SEA: SEA_Not_Null,
                PHASE: PHASE_Not_Null;
        end record;

--insert operation
        Procedure insert_domain_sea_phase(area_record: area_record_type);

--all gouped inserts must have non-null fields in a record so abstract interface operates
--on them as a record.

--update operations
        Procedure update_Area(area_record: area_record_type;
                                with_this_AREA_ID: AREA_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_Domain(area_record: area_record_type;
                                with_this_Domain: DOMAIN_Not_Null;
                                not_found· out boolean);
        Procedure update_SEA(area_record: area_record_type;
                                with_this_SEA: SEA_Not_Null;
                                not_found: out boolean);
        Procedure update_Phase(area_record: area_record_type;
                               with_this_Phase: PHASE_Not_Null;
                               not_found: out boolean);

--search operation
        Function search_domainseaphase(area_record: area_record_type) return boolean;

--delete operation
        Procedure delete_domain_sea_phase(area_record: area_record_type;
                                          deleted: out boolean);
--retrieve operation

        Function UniqueID return AREA_ID_Not_Null;


        Procedure get_area_record(for_area: AREA_ID_Not_Null;
                                  area_record: in out area_record_type;
                                  exists: out boolean);

end The_Area_Composite_OPS;
```

Package root_node_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Area_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Area_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types;

Package root_node_Composite_OPS is

        Type root_node_record_type is record
                AREA_ID: AREA_ID_Not_Null;
                GSC: GSC_ID_Not_Null;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_GCS_id_Area_id(root_node: root_node_record_type);

--update operations
--implemented by searched update
        Procedure update_Area(this_root_node: in root_node_record_type;
                              with_this_AREA_ID: AREA_ID_Not_Null;
                              not_found: out boolean);
        Procedure update_GSC(this_root_node: in root_node_record_type;
                              with_this_GSC_ID: GSC_ID_Not_Null;
                              not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_GCS_id_Area_id(root_node: root_node_record_type)
                                        return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_GCS_id_Area_id(root_node: root_node_record_type;
                                        is_deleted: boolean);
--retrieve operations
--implemented by cursor/select
        Package get_gsc_for_area is
                Procedure Open(for_this_AREA_ID: AREA_ID_Not_Null)
                Procedure Fetch(this_GSC_ID_record: in out root_node_record_type,
                                is_fetched: boolean)
                Procedure close;
        end get_gsc_for_area ;

        Package get_area_for_gsc is
                Procedure Open(for_this_GSC_ID: GSC_ID_Not_Null)
                Procedure Fetch(this_AREA_ID_record: in out root_node_record_type,
                                is_fetched: boolean)
                Procedure close;
        end get_area_for_gsc ;

end root_node_Composite_OPS;
```

Package General_Software_Characteristic_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        General_Software_Characteristic_primitive_domain_types,
        The_Area_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        General_Software_Characteristic_primitive_domain_types,
        The_Area_primitive_domain_types;

Package General_Software_Characteristic_Composite_OPS is

        Type GSC_record_type is record
                GSC: GSC_ID_Not_Null;
                CHARACT_NAME_Not_Null;
                formu_?_Type;
                evalu_?_Not_Null;
                evalu_help_Type;
                essential_flag_Type;
                evalu_method_Type;
                empirical_weight_Type;
        end record;


--insert operations
        Procedure insert_gsc_record(GSC_record: GSC_record_type);

--update operations
  --The null value is allowed to be input for all non-key attributes except evaluation
    --question since this question is mandatory for the Evaluator subsytem to prompt the
    --user for input
--Implemented by searched update
--
        Procedure update_gsc_id(for_given_GSC: GSC_ID_Not_Null;
                                GSC_ID: GSC_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_gsc_name(for_given_GSC: GSC_ID_Not_Null;
                                  name: CHARACT_NAME_Not_Null;
                                  not_found: out boolean);
        Procedure update_formulation_comment(for_given_GSC: GSC_ID_Not_Null;
                                             comment: formu_?_Type;
                                             not_found: out boolean);
        Procedure update_evaluation_question(for_given_GSC: GSC_ID_Not_Null;
                                             question: eval_?_Not_Null;
                                             not_found: out boolean);
        Procedure update_evaluation_help(for_given_GSC: GSC_ID_Not_Null;
                                         eval_help: evalu_help_Type;
                                         not_found: out boolean);
        Procedure update_essential_flag(for_given_GSC: GSC_ID_Not_Null;
                                        essential: essential_flag__Type;
                                        not_found: out boolean);
        Procedure update_evaluation_method(for_given_GSC: GSC_ID_Not_Null;
                                           eval_method: evalu_method_Type;
                                           not_found: out boolean);
        Procedure update_empirical_weight(for_given_GSC: GSC_ID_Not_Null;
                                          weight: empirical_weight_Type;
                                          not_found: out boolean);

--search operations

--delete operations
  -- all database delete operations work on row records.  To delete elements in a row
  --the update operation can be used with a null value.
        Procedure delete_gsc_record(for_given_GSC: GSC_ID_Not_Null;
                                    is_deleted: out boolean);
--retrieve operations
        Function UniqueID return GSC_ID_Not_Null;

        Procedure get_GSC_record_for_GSC (for_given_GSC: GSC_ID_Not_Null;
                                          GSC_record: in out GSC_record_type,
                                          exists: boolean);
        Package get_GSC_record_for_Area is
                Procedure Open(for_given_Area: Area_ID_Not_Null)
                Procedure Fetch(GSC_record: in out GSC_record_type,
                                   is_fetched: boolean)
                Procedure close;
        endget_GSC_record_for_Area;

end General_Software_Characteristic_Composite_OPS;
```

Package LinkGG_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        General_Software_Characteristic_primitive_domain_types,
        The_Area_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        General_Software_Characteristic_primitive_domain_types,
        The_Area_primitive_domain_types;

Package LinkGG_Composite_OPS is

        Type linkGG_record_type is record
                parent: GSC_ID_Not_Null;
                child: GSC_ID_Type;
        end record;

--insert operations
        Procedure insert_linkGG_record(linkGG_record: in out linkGG_record_type);
--Can't have a child without a parent but can have a parent that is childless
        Procedure insert_child_gsc(given_record: in out linkGG_record_type;
                                   insert_child: GSC_ID_Not_Null);

--update operations
        Procedure update_parent_gsc(given_record: in out linkGG_record_type;
                                   update_parent: GSC_ID_Not_Null;
                                   not_found: out boolean);
        Procedure update_child_gsc(given_record: in out linkGG_record_type;
                                   update_child: GSC_ID_Type;
                                   not_found: out boolean);

--search operations
        Procedure search_child_gsc(given_record: in out linkGG_record_type;
                                   exists: out boolean);
```

```
--delete operations
        Procedure delete_link(given_record: linkGG_record_type;
                                is_deleted: out boolean);
--retrieve
        Procedure get_Parent_Of(for_area: AREA_ID_Not_Null;
                                    for_child: GSC_ID_Not_Null;
                                    the_parent_record: in out linkGG_record_type);
        Package Get_children is
                Procedure Open(parent: GSC_ID_Not_Null;
                                  Area_ID: Area_ID_Not_Null);
                Procedure Fetch(record_of_child: in out linkGG_record_type,
                                  is_fetched: out boolean);
                Procedure close;
        end Get_children;

end LinkGG_Composite_OPS;
```

Package The_Tool_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Tool_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Tool_primitive_domain_types;

Package The_Tool_Composite_OPS is

        Type Tool_record_type is record
                TOOL_ID: TOOL_ID_Not_Null;
                TOOL_NAME: TOOL_NAME_Not_Null;
                VERSION: VERSION_Not_Null;
                vendor: vendor_Type;
                cost: cost_Type
        end record;

--insert operations
--implemented by insert values
        Procedure insert_tool_record(tool_record: Tool_record_type );

--update operations
--implemented by searched update
        Procedure update_Tool_ID(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                with_this_Tool_Id: TOOL_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_ToolName(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                with_this_ToolName: TOOL_NAME_Not_Null;
                                not_found: out boolean);
        Procedure update_Version(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                with_this_Version: VERSION_Not_Null;
                                not_found: out boolean);
        Procedure update_vendor(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                with_this_vendor: vendor_Type ;
                                not_found: out boolean);
        Procedure update_cost(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                with_this_cost: cost_Type  ;
                                not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_tool_record(tool_record: Tool_record_type)
                                        return boolean;

--delete operations
--implemented by searched delete
```

```
        Procedure delete_tool_record(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                     is_deleted: out boolean);
--retrieve operations
        Function UniqueID return TOOL_ID_Not_Null;
--implemented by cursor/select
        Procedure Tool_record_for_ID(for_this_TOOL_ID:TOOL_ID_Not_Null;
                                     this_Tool_record: in out Tool_record_type,
                                     exists: out boolean)


end The_Tool_Composite_OPS;
```

Package linkAT_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Tool_primitive_domain_types,
        The_Area_primitive_domain_types;

with    SQL_Base_Types_Pkg,
        The_Tool_primitive_domain_types,
        The_Area_primitive_domain_types;

Package linkAT_Composite_OPS is

        Type linkAT_record_type is record
                AREA_ID: AREA_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_LinkAT_record(linkAT_record: linkAT_record_type );

--update operations
--implemented by searched update
        Procedure update_Area(linkAT_record: linkAT_record_type ;
                                with_this_AREA_ID: AREA_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_Tool(Tool_ID: TOOL_ID_Not_Null;
                                with_this_TOOL_ID: TOOL_ID_Not_Null;
                                not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_linkAT_record(linkAT_record: linkAT_record_type )
                                        return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_linkAT_record(linkAT_record: linkAT_record_type;
                                        is_deleted: out boolean);
--retrieve operations
--implemented by cursor/select
        Package get_tools_for_area is
                Procedure Open(for_this_AREA_ID: AREA_ID_Not_Null)
                Procedure Fetch(this_TOOL_ID_record: in out linkAT_record_type,
                                is_fetched: out boolean)
                Procedure close;
        end get_tools_for_area ;

        Package get_areas_for_tool is
                Procedure Open(for_this_TOOL_ID: TOOL_ID_Not_Null)
                Procedure Fetch(this_AREA_ID_record: in out linkAT_record_type,
                                is_fetched: out boolean)
                Procedure close;
        end get_areas_for_tool ;

end linkAT_Composite_OPS;
```

Package The_Evaluator_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Evaluator_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Evaluator_primitive_domain_types;

Package The_Evaluator_Composite_OPS is

        Type Evaluator_record_type is record
                EVAL_ID: EVAL_ID_Not_Null;
                FIRST_NAME: FIRST_NAME_Not_Null;
                LAST_NAME: LAST_NAME_Not_Null;
                date: date_Type;
                type: type_Type
        end record;

--insert operations
--implemented by insert values
        Procedure insert_Evaluator_record(Evaluator_record: Evaluator_record_type );

--update operations
--implemented by searched update
        Procedure update_EVAL_ID(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                with_this_Evaluator_Id: EVAL_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_FIRST_NAME(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                with_this_First_Name: FIRST_NAME_Not_Null;
                                not_found: out boolean);
        Procedure update_LAST_NAME(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                with_this_LAST_NAME: LAST_NAME_Not_Null;
                                not_found: out boolean);
        Procedure update_date(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                with_this_date: date_Type ;
                                not_found: out boolean);
        Procedure update_type(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                with_this_type: type_Type  ;
                                not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_Evaluator_record(Evaluator_record: Evaluator_record_type)
                                    return boolean;

--delete operations
--implemented by searched delete
```

```
        Procedure delete_Evaluator_record(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                        is_deleted: out boolean);
--retrieve operations
        Function UniqueID return EVAL_ID_Not_Null;
--implemented by cursor/select
        Procedure Evaluator_record_for_ID(for_this_EVAL_ID:EVAL_ID_Not_Null;
                                             this_Evaluator_record:
                                             in out Evaluator_record_type,
                                             exists: out boolean)

end The_Evaluator_Composite_OPS;
```

Package linkES_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Evaluator_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;



use     SQL_Base_Types_Pkg,
        The_Evaluator_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;

Package linkES_Composite_OPS is

        Type linkES_record_type is record
                EVAL_ID: EVAL_ID_Not_Null;
                SSC_ID: SSC_ID_Not_Null;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_LinkES_record(linkES_record: linkES_record_type );

--update operations
--implemented by searched update
        Procedure update_Eval_ID(linkES_record: linkES_record_type ;
                                 with_this_EVAL_ID: EVAL_ID_Not_Null;
                                 not_found: out boolean);
        Procedure update_SSC_ID(linkES_record: linkES_record_type ;
                                 with_this_SSC_ID: SSC_ID_Not_Null;
                                 not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_linkES_record(linkES_record: linkES_record_type )
                                         return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_linkES_record(linkES_record: linkES_record_type;
                                        is_deleted: out boolean);
--retrieve operations
--implemented by cursor/select
        Package get_ssc_ids_for_eval_ID is
                Procedure Open(for_this_EVAL_ID: EVAL_ID_Not_Null);
                Procedure Fetch(this_SSC_ID_record: in out linkES_record_type,
                                  is_fetched: out boolean);
                Procedure close;
        end get_ssc_ids_for_eval_ID ;

        Package get_eval_ids_for_ssc_id is
                Procedure Open(for_this_SSC_ID: SSC_ID_Not_Null);
                Procedure Fetch(this_EVAL_ID_record: in out linkES_record_type,
                                  is_fetched: out boolean);
                Procedure close;
        end get_eval_ids_for_ssc_id;


end linkES_Composite_OPS;
```

Package The_Quality_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Quality_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Quality_primitive_domain_types;

Package The_Quality_Composite_OPS is

        Type Quality_record_type is record
                QUAL_ID: QUAL_ID_Not_Null;
                QUALITY_NAME: QUALITY_NAME_Not_Null;
                QUALITY_VALUE: QUALITY_VALUE_Not_Null;
        end record;

--insert operations
--implemented by insert values
        Procedure insert_Quality_record(Quality_record: Quality_record_type );

--update operations
--implemented by searched update
        Procedure update_QUAL_ID(for_this_QUAL_ID:QUAL_ID_Not_Null;
                                with_this_Quality_Id: QUAL_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_QUALITY_NAME(for_this_QUAL_ID:QUAL_ID_Not_Null;
                                        with_this_QUALITY_NAME:
                                        QUALITY_NAME_Not_Null;
                                        not_found: out boolean);
        Procedure update_QUALITY_VALUE
                                        (for_this_QUAL_ID:QUAL_ID_Not_Null;
                                        with_this_QUALITY_VALUE:
                                        QUALITY_VALUE_Not_Null;
                                        not_found: out boolean);
```

```
--search operation
--implemented by select,fetch, check
        Function search_Quality_record(for_this_QUAL_ID:QUAL_ID_Not_Null;
                                        return boolean;

--delete operations
--implemented by searched delete
        Procedure delete_Quality_record(Quality_record: Quality_record_type
                                        is_deleted: out boolean);
--retrieve operations
        Function UniqueID return QUAL_ID_Not_Null;
--implemented by select/cursor select
        Procedure Quality_record_for_ID(for_this_QUAL_ID:QUAL_ID_Not_Null;
                                                this_Quality_record:
                                                in out Quality_record_type,
                                                exists: out boolean);


end The_Quality_Composite_OPS;
```

Package linkQS_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Quality_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Quality_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;

Package linkQS_Composite_OPS is

        Type linkQS_record_type is record
                QUAL_ID: QUAL_ID_Not_Null;
                SSC_ID: SSC_ID_Not_Null;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_LinkQS_record(linkQS_record: linkQS_record_type );

--update operations
--implemented by searched update
        Procedure update_QUAL_ID(for_this_QUAL_ID: QUAL_ID_Not_Null;
                                 with_this_QUAL_ID: QUAL_ID_Not_Null;
                                 not_found: out boolean);
        Procedure update_SSC_ID(for_this_QUAL_ID: QUAL_ID_Not_Null;
                                with_this_SSC_ID: SSC_ID_Not_Null;
                                not_found: out boolean);
```

```
--search operation
--implemented by select,fetch, check
        Function search_linkQS_record(linkQS_record: linkQS_record_type )
                                            return boolean;
--delete operations
--implemented by searched delete
        Procedure delete_linkQS_record(for_this_QUAL_ID: QUAL_ID_Not_Null;
                                            is_deleted: boolean);
--retrieve operations
--implemented by cursor/select
        Package get_SSC_IDs_for_QUAL_ID is
                Procedure Open(for_this_QUAL_ID: QUAL_ID_Not_Null);
                Procedure Fetch(this_SSC_ID_record: in out linkQS_record_type,
                                is_fetched: out boolean);
                Procedure close;
        end get_SSC_IDs_for_QUAL_ID ;

        Package get_QUAL_IDs_for_SSC_ID is
                Procedure Open(for_this_SSC_ID: SSC_ID_Not_Null);
                Procedure Fetch(this_QUAL_ID_record: in out linkQS_record_type,
                                is_fetched: out boolean);
                Procedure close;
        end get_QUAL_IDs_for_SSC_ID ;


end linkQS_Composite_OPS;
```

Package The_Specific_Software_Characteristic_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        The_Specific_Software_Characteristic_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types,
        The_Tool_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        The_Specific_Software_Characteristic_primitive_domain_types
        General_Software_Characteristic_primitive_domain_types,
        The_Tool_primitive_domain_types;

Package The_Specific_Software_Characteristic_Composite_OPS is

        Type SSC_record_type is record
                TOOL_ID: TOOL_ID_Not_Null;
                GSC_ID: GSC_ID_Not_Null;
                SSC_ID: SSC_ID_Not_Null;
                value: value_Type;
                tep: tep_Type
        end record;

--insert operations
--implemented by insert values
        Procedure insert_ SSC_record(SSC_record: SSC_record_type );

--update operations
--implemented by searched update
        Procedure update_Tool_ID(for_this_SSC_ID:SSC_ID_Not_Null;
                                 with_this_Tool_Id: TOOL_ID_Not_Null;
                                 not_found: out boolean);
        Procedure update_GSC_ID(for_this_SSC_ID:SSC_ID_Not_Null;
                                 with_this_GSC_ID: GSC_ID_Not_Null;
                                 not_found: out boolean);
        Procedure update_SSC_ID(for_this_SSC_ID:SSC_ID_Not_Null;
                                 with_this_SSC_ID: SSC_ID_Not_Null;
                                 not_found: out boolean);
        Procedure update_value(for_this_SSC_ID:SSC_ID_Not_Null;
                                 with_this_value: value_Type ;
                                 not_found: out boolean);
        Procedure update_tep(for_this_SSC_ID:SSC_ID_Not_Null;
                                 with_this_tep: tep_Type  ;
                                 not_found: out boolean);

--search operation
--implemented by select,fetch, check
        Function search_SSC_record(SSC_record: SSC_record_type)
                                        return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_SSC_record(for_this_SSC_ID:SSC_ID_Not_Null;
                                    is_deleted: out boolean);
--retrieve operations
        Function UniqueID return SSC_ID_Not_Null;
--implemented by cursor/select
        Procedure SSC_record_for_ID(for_this_SSC_ID:SSC_ID_Not_Null;
                                    this_SSC_record: in out SSC_record_type,
                                    exists: out boolean)
        Function UniqueID return SSC_ID_Not_Null;



end The_Specific_Software_Characteristic_Composite_OPS;
```

Package Weight_Set_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        Weight_Set_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        Weight_Set_primitive_domain_types;

Package Weight_Set_Composite_OPS is

        Type Weight_set_record_type is record
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                default: default_Type;
        end record;

--insert operations
--implemented by insert values
      Procedure insert_Weight_set_record(Weight_set_record: Weight_set_record_type );

--update operations
--implemented by searched update
        Procedure update_WEIGHT_SET_NAME
                (for_this_WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_Weight_set_Id: WEIGHT_SET_NAME_Not_Null;
                not_found: out boolean);
        Procedure update_default
                (for_this_WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_default: default_Type;
                not_found: out boolean);
--search operation
--implemented by select,fetch, check
        Function search_Weight_set_record
                (Weight_set_record: Weight_set_record_type) return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_Weight_set_record
                (Weight_set_record: Weight_set_record_type
                is_deleted: out boolean);
--retrieve operations
--implemented by cursor/select
        Procedure Weight_set_record_for_name
                (for_this_WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                this_Weight_set_record: in out Weight_set_record_type,
                exists: out boolean);



end Weight_Set_Composite_OPS;
```

Package Selection_Set_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        Selection_Set_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        Selection_Set_primitive_domain_types;

Package Selection_Set_Composite_OPS is

        Type Selection_Set_record_type is record
                SET_NAME: SET_NAME_Not_Null;
        end record;

--insert operations
--implemented by insert values
        Procedure insert_Selection_Set_record(Selection_Set_record:
                                                  Selection_Set_record_type );

--update operations
--implemented by searched update
        Procedure update_SET_NAME(Selection_Set_record:
                                          Selection_Set_record_type;
                                          with_this_Selection_Set_Name:
                                          SET_NAME_Not_Null;
                                           not_found: out boolean);
--search operation
--implemented by select,fetch, check
        Function search_Selection_Set_record(Selection_Set_record:
                                  Selection_Set_record_type) return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_Selection_Set_record(Selection_Set_record:
                                              Selection_Set_record_type
                                              is_deleted: out boolean);
--retrieve operations
--implemented by cursor/select
        Procedure Selection_Set_record_for_name
                (for_this_SET_NAME: SET_NAME_Not_Null;
                this_Selection_Set_record:
                in out Selection_Set_record_type,
                exists: out boolean);

end Selection_Set_Composite_OPS;
```

Package linkSAWT_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        Selection_Set_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        The_Tool_primitive_domain_types,
        The_Area_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        Selection_Set_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        The_Tool_primitive_domain_types,
        The_Area_primitive_domain_types;

Package linkSAWT_Composite_OPS is

        Type linkSAWT_record_type is record
                SET_NAME: SET_NAME_Not_Null;
                AREA_ID: AREA_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_LinkSAWT_record
                (linkSAWT_record: linkSAWT_record_type );

--update operations
--implemented by searched update
        Procedure SET_NAME(linkSAWT_record: linkSAWT_record_type ;
                                with_this_SET_NAME: SET_NAME_Not_Null;
                                not_found: out boolean);
        Procedure update_Area(linkSAWT_record: linkSAWT_record_type ;
                                with_this_AREA_ID: AREA_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_Tool(linkSAWT_record: linkSAWT_record_type ;
                                with_this_TOOL_ID: TOOL_ID_Not_Null;
                                not_found: out boolean);
        Procedure update_WEIGHT_SET_NAME
                                    (linkSAWT_record: linkSAWT_record_type ;
                                    with_this_WEIGHT_SET_NAME:
                                    WEIGHT_SET_NAME_Not_Null;
                                    not_found: out boolean);

--search operation
--implemented by select,fetch, check
```

```
          Function search_linkSAWT_record(linkSAWT_record: linkSAWT_record_type )
                                                  return boolean;

--delete operations
--implemented by searched delete
          Procedure delete_linkSAWT_record(linkSAWT_record: linkSAWT_record_type;
                                                  is_deleted: out boolean);
--retrieve operations
--implemented by cursor/select
          Package get_tools_for_SET_NAME is
                    Procedure Open(for_this_SET_NAME: SET_NAME_Not_Null);
                    Procedure Fetch(linkSAWT_record: in out linkSAWT_record_type,
                                        is_fetched: out boolean);
                    Procedure close;
          end get_tools_for_SET_NAME ;

          Package get_SET_NAMES_for_tool is
                    Procedure Open(for_this_TOOL_ID: TOOL_ID_Not_Null);
                    Procedure Fetch(linkSAWT_record: in out linkSAWT_record_type,,
                                        is_fetched: out boolean);
                    Procedure close;
          end get_SET_NAMES_for_tool;

end linkSAWT_Composite_OPS;
```

Package software_char_score_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        software_char_score_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        software_char_score_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        The_Specific_Software_Characteristic_primitive_domain_types;

Package software_char_score_Composite_OPS is

        Type software_char_score_record_type is record
                SSC_ID: SSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                function_score: function_score_Type;
                quality_score: quality_score_Type;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_software_char_score_record
                (software_char_score_record: software_char_score_record_type );
```

```
--update operations
--implemented by searched update
        Procedure SSC_ID
                (SSC_ID: SSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null
                with_this_SSC_ID: SSC_ID_Not_Null;
                not_found: out boolean);
        Procedure update_function_score
                (SSC_ID: SSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null
                with_this_function_score: function_score_Type;
                not_found: out boolean);
        Procedure update_quality_score
                (SSC_ID: SSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null
                with_this_quality_score: quality_score_Type;
                not_found: out boolean);
        Procedure update_WEIGHT_SET_NAME
                        (SSC_ID: SSC_ID_Not_Null;
                        WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null
                        with_this_WEIGHT_SET_NAME:
                        WEIGHT_SET_NAME_Not_Null;
                        not_found: out boolean);


--search operation
--implemented by select,fetch, check
        Function search_software_char_score_record
                        (SSC_ID: SSC_ID_Not_Null;
                        WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null)
                        return boolean;
```

```
--delete operations
--implemented by searched delete
        Procedure delete_software_char_score_record
                (SSC_ID: SSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                is_deleted: out boolean);
--retrieve operations
--implemented by select/cursor select
        Procedure get_scores
                        (SSC_ID: SSC_ID_Not_Null;
                        WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                        software_char_score_record:
                        in out software_char_score_record_type;
                        is_fetched: out boolean);


end software_char_score_Composite_OPS;
```

Package tool_score_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        tool_score_primitive_domain_types,
        The_Tool_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        tool_score_primitive_domain_types,
        The_Tool_primitive_domain_types,
        Weight_Set_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types;

Package tool_score_Composite_OPS is

        Type tool_score_record_type is record
                GSC_ID: GSC_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                function_score: function_score_Type;
                quality_score: quality_score_Type;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_tool_score_record
                (tool_score_record: tool_score_record_type );
```

```
--update operations
--implemented by searched update
        Procedure update_GSC_ID(  GSC_ID: GSC_ID_Not_Null;
                                  TOOL_ID: TOOL_ID_Not_Null;
                                  WEIGHT_SET_NAME:
                                  WEIGHT_SET_NAME_Not_Null;
                                  with_this_GSC_ID: GSC_ID_Not_Null;
                                  not_found: out boolean);
        Procedure update_TOOL_ID(GSC_ID: GSC_ID_Not_Null;
                                  TOOL_ID: TOOL_ID_Not_Null;
                                  WEIGHT_SET_NAME:
                                  WEIGHT_SET_NAME_Not_Null;
                                  with_this_TOOL_ID: TOOL_ID_Not_Null;
                                  not_found: out boolean);
        Procedure update_function_score
                (GSC_ID: GSC_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
                WEIGHT_SET_NAME:
                WEIGHT_SET_NAME_Not_Null;
                with_this_function_score: function_score_Type;
                not_found: out boolean);
        Procedure update_quality_score
                (GSC_ID: GSC_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
                WEIGHT_SET_NAME:
                WEIGHT_SET_NAME_Not_Null;
                with_this_quality_score: quality_score_Type;
                not_found: out boolean);
        Procedure update_WEIGHT_SET_NAME
                (GSC_ID: GSC_ID_Not_Null;
                TOOL_ID: TOOL_ID_Not_Null;
                WEIGHT_SET_NAME:
                WEIGHT_SET_NAME_Not_Null;
                with_this_WEIGHT_SET_NAME:
                WEIGHT_SET_NAME_Not_Null;
                not_found: out boolean);


--search operatio
--implemented by select,fetch, check
        Function search_tool_score_record
                    (tool_score_record: tool_score_record_type)
                    return boolean;

--delete operations
--implemented by searched delete
```

```
          Procedure delete_tool_score_record
                   (GSC_ID: GSC_ID_Not_Null;
                   TOOL_ID: TOOL_ID_Not_Null;
                   WEIGHT_SET_NAME:
                   WEIGHT_SET_NAME_Not_Null;
                   is_deleted: out boolean);
--retrieve operations
--implemented by select/cursor select
          Procedure get_scores
                   (GSC_ID: GSC_ID_Not_Null;
                   TOOL_ID: TOOL_ID_Not_Null;
                   WEIGHT_SET_NAME:
                   WEIGHT_SET_NAME_Not_Null;
                   tool_score_record:
                   in out tool_score_record_type;
                   is_fetched: out boolean);


end tool_score_Composite_OPS;
```

Package software_char_weight_Composite_OPS

```
with    SQL_Base_Types_Pkg,
        software_char_weight_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types
        Weight_Set_primitive_domain_types;

use     SQL_Base_Types_Pkg,
        software_char_weight_primitive_domain_types,
        General_Software_Characteristic_primitive_domain_types
        Weight_Set_primitive_domain_types;

Package software_char_weight_Composite_OPS is

        Type software_char_weight_record_type is record
                GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                function_weight: function_weight_Type;
                quality_weight: quality_weight_Type;
        end record;


--insert operations
--implemented by insert values
        Procedure insert_software_char_weight_record
                (software_char_weight_record: software_char_weight_record_type );

--update operations
--implemented by searched update
        Procedure GSC_ID
                (GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_GSC_ID: GSC_ID_Not_Null;
                out_found: out boolean);
        Procedure update_function_weight
                (GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_function_weight: function_weight_Type;
                not_found: out boolean);
        Procedure update_quality_weight
                (GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_quality_weight: quality_weight_Type;
                not_found: out boolean);
        Procedure update_WEIGHT_SET_NAME
                (GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                with_this_WEIGHT_SET_NAME:
                WEIGHT_SET_NAME_Not_Null;
                not_found: out boolean);


--search operation
--implemented by select,fetch, check
        Function search_software_char_weight_record
                        (GSC_ID: GSC_ID_Not_Null;
                        WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null)
                        return boolean;

--delete operations
--implemented by searched delete
        Procedure delete_software_char_weight_record
                (GSC_ID: GSC_ID_Not_Null;
                WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                is_deleted: out boolean);
--retrieve operations
--implemented by select/cursor select
        Procedure get_weights
                        (GSC_ID: GSC_ID_Not_Null;
                        WEIGHT_SET_NAME: WEIGHT_SET_NAME_Not_Null;
                        software_char_weight_record:
                        in out software_char_weight_record_type;
                        is_fetched: out boolean);
```

```
          Package get_weight_Sets_for_GSC_ID is
                  Procedure Open(GSC_ID: GSC_ID_Not_Null);
                  Procedure Fetch(software_char_weight_record:
                                          in out software_char_weight_record_type;
                                          is_fetched: out boolean);
                  Procedure close;
          end get_weight_Sets_for_GSC_ID;

          Package get_GSC_IDS_for_Weight_Set is
                  Procedure Open
                          (for_this_Weight_Set: WEIGHT_SET_NAME:
                          WEIGHT_SET_NAME_Not_Null);
                  Procedure Fetch(software_char_weight_record:
                                          in out software_char_weight_record_type;
                                          is_fetched: out boolean);
                  Procedure close;
          end get_GSC_IDS_for_Weight_Set ;


  end software_char_weight_Composite_OPS;
```

References

1. *Draft Recommended Practice for the Evaluation and Selection of CASE Tools,* Computer Society of the IEEE, January 1991.

2. *E&V Guidebook,* Ada Joint Program Office, February 1991.

3. *Notes from CASE Managment Workshop,* Carnegie Mellon University, June 1991.

4. *Congressional Mandate for Ada,* Software Technology Support Center CrossTalk, Hill AFB Utah, May 1991, Public Law 101-511, signed by the President, November 1990 .

5. *CSCE 645 Database Couse Notes,* 1991.

6. *Consultation with Air Force Institute of Technology Database Instructor, Dr. Mark A. Roth,* September 1991.

7. Alderucci, D., *Evaluator, version 1.0,* Charles Stark Draper Laboratoriy, Inc, February 1991.

8. Alderucci, D., *Formualor, version 1.0,* Charles Stark Draper Laboratoriy, Inc, February 1991.

9. Alderucci, D., *Selector, version 1.0,* Charles Stark Draper Laboratoriy, Inc, February 1991.

10. Batt, G.T. *CASE Technology and the Systems Developement Life Cycle: a proposed integration of CASE tools with DoD STD-2167A.* Master's thesis, Naval Post Graduate School, 1989 (AD-A207 844).

11. Bigelow, J. "Hypertext and CASE." *IEEE Software*: 23-27 (March 1988).

12. Brooks, F.P. *The Mythical Man-Month.* Reading, MA: Addison-Wesley Publishing Company 1982.

13. Chastek, G.J., M.H. Graham, and G. Zelesnik "The SQL Ada Module Description Language SAMeDL." Technical Report. CMU/SEI-90-TR-26. Software Engineering Institute, November 1990.

14. Chastek, G.J., M.H. Graham, and G. Zelesnik "Rational for SQL Ada Module Description Language SAMeDL." Technical Report. CMU/SEI-91-TR-4. Software Engineering Institute, March 1991.

15. Date, C.J. *A Guide to THE SQL STANDARD.* Addison-Wesley Publishing Company 1987.

16. Engle, C., R. Firth, M.H. Graham, and W.G. Wood "Interfacing Ada and SQL ." Technical Report. CMU/SEI-87-TR-48. Software Engineering Institute, December 1987.

17. Graham, M.H. "Guidelines for the Use of SAME." Technical Report. CMU/SEI-89-TR-16. Software Engineering Institute, May 1989.

18. Hanrahan, B., J.V. Buren, C. Rieping, T. Fujita-Yuhas, J. Grotzky, G. Jones, J. Petersen, and G. Peterson "Requirements Analysis & Design Tool Report." Technical Report. Software Technology Support Center, Hill Air Force Base UT, 1991.

19. Hildebrant, R.R. "Requirements, Design, and Usage of stemDB, a Software Test and Evaluation Database." Technical Report. Charles Stark Draper Laboratoriy, Inc, Cambridge MA, March 1991.

20. Hughes, C.T. and J.D.C. "The Stages of CASE Usage." *Datamation*: 41-44 (February 1990).

21. Humphrey, W.S. *Managing the Software Process*. Addison-Wesley Publishing Company 1990.

22. Keuffel, W. "CASE for the rest of us." *Computer Language*: 25-29 (1991).

23. Korth, H.F. and A. Silberschatz *Database System Concepts,* McGraw-Hill, Inc. (1991), pp. 24-213.

24. Lawlis, P.K. *Supporting Selection Decisions Based on The Technical Evaluation of Ada Environments and Their Components*. Ph.D. dissertation, Arizona State University, August 1989.

25. *Oracle for Macintosh, Reference, Version 1.2*. Oracle Corporation, 1990.

26. Osterweil, L. "Software Environment Research, Directions for the Next Five Years." *IEEE Computer*: 35-43 (1981).

27. Petersen, G., G. Daich, D. Dyer, and S. Atkinson " Interim Report on Requirements Analysis and Design Tools." Technical Report. Software Technology Support Center, Hill Air Force Base UT, November 1990.

28. Sims, M.L. *A Review of the suitability of Available Computer Aided Software Engineering (CASE) Tools for the Small Software Development Environment.* Master's thesis, AFIT/CI/CIA-89-077. University of South Florida, 1988.

29. Tamanaha, D.Y. "The Application of CASE in Large Aerospace Projects." *1989 IEEE Aerospace Applications Conference digest*: 1-18 (1989).

30. Wybolt, N. "Perspectives on CASE Tool Integration." In *Software Engineering Notes*. ACM Press, pp. 56-60, 1991.

Vita

Captain Tina M. DeAngelis was born on 26 February 1964 in Watertown, Massachusetts. After graduating from high school she attended Purdue University in West Lafayette, Indiana. She received a three year ROTC scholarship and after receiving both her bachelors degree in Electrical Engineering and her Air Force Commission was assigned to Material Management (MM), Hill AFB, Utah. She started her job at Hill as a member of a team that was responsible for developing a new Navigation and Weapon Delivery System computer software and hardware for the F4 Phantom. Her duties included being involved in the flight testing of the ongoing development effort. Just before leaving Hill for AFIT, she was reassigned to another computer hardware and software development effort (the VHSIC Core Avionics Processor program (VCAP)) for the F-16A/B Falcon flight computer. She entered the School of Engineering, Air Force Institute of Technology in May of 1990. Her current address is Space Systems Division, L. A. California.

Permanent address: 49 Hazel St
Watertown, Ma 02172

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1991	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE ANALYSIS OF A DECISION SUPPORT SYSTEM FOR CASE TOOL SELECTION AND THE SPECIFICATION OF AN ADA TO SQL ABSTRACT INTERFACE	5. FUNDING NUMBERS
6. AUTHOR(S) Tina M. DeAngelis, Capt, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/91D-04
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Software Technology Support Center (STSC) Ogden ALC/TISAC Hill AFB, Utah 84056	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution unlimitted	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Information overload has long been a problem in the fast moving technical field of software development. Yet quality information is needed to make informed decisions about buying software tools that help in software development. Computer Aided Software Engineering (CASE) tools help to coordinate and control information in large software developments. Many CASE tool purchases, however, are being based on ad hoc tool evaluation and selection methods which depend on biased vendor information. To capture specific knowledge about how to pick a tool for a given software development effort. a historical database that identifies important tool characteristics needed to be maintained by an unbiased organization and a mechanism (in the form of a decision support system) for interpreting that database needed to be made available.

To address this deficiency, the Software Technology Support Center at Hill AFB in Utah was developing a CASE tool selection support tool, the STEMdB. This research accomplishes an analysis of this tool and suggests ways to make it more robust, portable and maintainable. It presents an object oriented approach to the design while addressing the issue of portability by accomplishing an Ada to Structured Query Language (SQL) abstract interface.

14. SUBJECT TERMS Ada interface. SQL interface. Decision Support System. CASE tool selection. SQL. Ada, Ada to SQL binding	15. NUMBER OF PAGES 206
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102