Semi-Automatic Development of Payload Operations Control Center Software

Contract Number NAS5-31500
Task 28-11600

Prepared For:

Computer Sciences Corporation
8728 Colesville Road
Silver Spring, MD 20910

October 31, 1988

CTA INCORPORATED
14900 Sweitzer Lane, Suite 201
Laurel, MD 20707
(301)369-2400

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AP	Application Processor
CAP	Command Acceptance Pattern
COBE	Cosmic Background Explorer
CDOS	Customer Data Operations System
CMS	Command Management System
DE	Dynamic Explorer
DOCS	Data Operations Control System
ERBS	Earth Radiation Budget Satellite
FDF	Flight Dynamics Facility
GMT	Greenwich Mean Time
GRO	Gamma Ray Observatory
I/O	Input/Output
ISEE	International Sun/Earth Explorer
MAE	MSOCC Applications Executive
MODLAN	Mission Operations Division Local Area Network
MSOCC	Multi-Satellite Operations Control Center
NASCOM	NASA Communications Network
NCC	Network Communications Center
OOD	Object-Oriented Design
OBC	On-board Computer
ODB	Operational Database
ODN	Operations Data Network
POCC	Payload Operations Control Center
PDS	Program Design Specification
SRS	Software Requirements Specification
STARS	Software Technology for Adaptible, Reliable Systems
STDN	Space Tracking and Data Network
STOL	Standard Test and Operations Language
TAC	Telemetry and Command
TDRSS	Tracking and Data Relay Satellite System
TUT	Telemetry Users Table
VHLL	Very High Level Language

1 INTRODUCTION

This report summarizes the current status of CTA's investigation of methods and tools for automating the software development process in NASA Goddard Space Flight Center, Code 500 (see references [1], [2], and [3] for background). The emphasis in this effort has been on methods and tools in support of software reuse. The most recent phase of the effort has been a domain analysis of Payload Operations Control Center (POCC) software. This report summarizes the results of the domain analysis, and proposes an approach to semi-automatic development of POCC Application Processor (AP) software based on these results.

The domain analysis enabled us to abstract, from specific systems, the typical components of a POCC AP. We were also able to identify patterns in the way one AP might be different from another. These two perspectives--aspects that tend to change from AP to AP, and aspects that tend to remain the same--suggest an overall approach to the reuse of POCC AP software.

We found that different parts of an AP require different development technologies. We propose a hybrid approach that combines *constructive* and *generative* technologies. Constructive methods emphasize the assembly of pre-defined reusable components. Generative methods provide for automated generation of software from specifications in a very-high-level language (VHLL). Table 1-1 presents the relevant technologies. In the next phase of our effort, we propose to demonstrate how these technologies can be combined to facilitate AP development.

1.1 Outline

Section Two presents the results of the domain analysis in terms of the stable and variable properties of AP software. Based on these properties, we identify three principal obstacles to reuse: 1) mission-specific requirements, 2) obsolesence of hosts and external interfaces, and 3) increasing complexity of requirements. We propose a technical approach to overcoming each of these obstacles.

Section Three describes the proposed technical approach in more detail. We describe how we would enrich the prototype Software Reuse Environment [4] to include the reusable components identified during the domain analysis. We also describe a possible scenario of semi-automatic development of an AP, illustrating how the proposed technologies would work together to support system development. The scenario may be considered as a refinement of our operational concept of software reuse, which was developed in an earlier phase of this effort [5].

In Section Four we summarize our concept of semi-automatic development, and outline the objectives for the next phase of this effort.

Table 1-1: POCC Software Development Requires a Combination of Constructive and Generative Techniques

Constructive techniques:

- *Dialog-based specification:* Automated assembly of components, based on question-and-answer interaction with the developer. Suitable for configuring the top level of an AP.
- *Interactive selection of components:* Navigation of available components in the Reuse Database. Suitable for configuring standard interfaces (such as Operator Input/Output) and for selecting standard processing functions (such as Telemetry processing)

Generative techniques:

- *Domain-specific Very-High-Level Language (VHLL).* Suitable for generating the AP database together with those components that access the database
- *Graphical programming:* automated source code generation from diagrams. Suitable for assembling new combinations of interactively selected components.

2 SUMMARY OF DOMAIN ANALYSIS

2.1 Scope of the Analysis

We examined the requirements specifications of seven systems developed for the Multi-Satellite Operations Control Center (MSOCC):

- o Standard Software
- o Dynamic Explorer (DE)
- o International Sun/Earth Explorer (ISEE)
- o Earth Radiation Budget Satellite (ERBS)
- o MSOCC Applications Executive (MAE)
- o Gamma Ray Observatory (GRO)
- o Cosmic Background Explorer (COBE)

Standard Software and MAE are both baselines for POCC APs. The APs for specific missions are built by adding to, modifying, and/or replacing parts of the baseline software. DE, ISEE, and ERBS are based upon Standard Software, while GRO and COBE are based upon MAE.

We chose these systems because 1) both Standard Software and MAE are based on analyses of the reusable aspects of POCC APs, and 2) the APs that have been built on these baselines provide lessons about what can, and what cannot, be reused. Including Standard Software and MAE enabled us to take advantage of the analyses that went into these systems. Including APs that are built on each of the baselines enabled us to evaluate the success of the attempted reuse.

We restricted our study to high-level specifications of these systems (references [6] through [12]). We used the Software Requirements Specifications (SRSs) of ERBS, MAE, COBE, and GRO, the Final Report of Standard Software, and the Program Design Specifications (PDSs) of ISEE and DE. The emphasis on high-level documentation was in part due to limitations of available resources for the project, but there are also intrinsic justifications for the restriction. To understand a domain, we must first and foremost understand the requirements. We cannot expect to identify all possible areas of reuse without understanding the requirements, because similar requirements may be realized by apparently dissimilar software designs and implementations.

Concentrating on requirements has, in addition, forced us to examine high-level components. In recommending steps toward automating software development in Code 500, we predicted that the greatest leverage would come from reuse of high-level components. Our advocacy of object-oriented design (OOD) was based, in part, on the fact that OOD facilitates reuse of high-level components by establishing unambiguous interfaces and control relationships [3]. While the subsystems of all the APs we studied are similar in name and overall function, reusing any of them individually would be difficult because the interfaces are not formulated in object-oriented terms. We expect

the benefit of OOD to become apparent when we develop an object-oriented interface to each of the standard subsystems of a POCC AP.

2.2 Results of the Analysis

The similarities and differences between the systems we studied are documented in Section Three of reference [13]. Table 2-1 classifies the variations we found into six types. Identifying these types of variation enabled us to identify the obstacles to reuse, and to develop approaches to overcoming each obstacle.

We found that POCC APs are typically composed of the subsystems shown in Table 2-2. The overall function of each subsystem is fairly constant across APs. As illustrated in Table 2-3a, however, the exact partition into subsystems may vary across APs.

At the level immediately below subsystems, we find some components that are *optional* but *stable*. These components may or may not be present in an AP. When they are present, however, they are likely to require only minimal, if any, tailoring for the specific mission. Table 2-3b lists some of the components in this category.

A slightly different category consists of components that are not optional, but for which a range of stable options exists. Every AP, for example, must contain an operator interface through which directives can be entered. The Standard Test and Operations Language (STOL) interpreter is a standard solution to this requirement, but it is not the only possible solution. Future APs may employ operator interfaces based on modern graphics hardware. Table 2-3c contains some other examples in this category.

The internal functions and structures of some of the subsystems are more difficult to classify. We found variations in the specific functions performed by a subsystem, in the way these functions are ordered, and in the granularity with which low-level functions are grouped into higher-level functions. Telemetry subsystems, for example, all contain certain standard functions, such as limit checking, stripping of data, and retrieval of Onboard Computer (OBC) dumps. The details of how these functions are performed, however, are tied to the specific mission.

Many of these differences are reflected in the schema of the AP database. Database fields define the limits to be checked, the time intervals at which data are to be sampled, and other parameters of telemetry processing (as well as of command, display, and STOL processing). These fields are treated parametrically within a mission, but apparently not across missions. Parameterization across missions represents one strategy that could be employed to increase reuse of POCC AP software. A technical approach to such parameterization is proposed in Section 2.3.1.

Some functions, such as scientific processing in the Telemetry subsystem, will necessarily be unique to a mission. Even these, however, will reuse certain standard low-level functions. In the case of scientific processing, a library of standard mathematical functions will typically be used. Our perception is that this level of reuse is already occurring in POCC AP development. Further progress towards automating development can be achieved through tools that facilitate the composition of low-level reusable functions into mission-specific higher-level functions. Section 2.3.3 describes one possible approach.

Table 2-1: Variations Across APs can be Classified into Six Types

- o External interfaces
- o Devices supported
- o Data definitions
- o Processing performed
- o Placement of functions
- o Leveling

Table 2.2: POCC APs Typically Contain A Small Number of Standard Subsystems

- *External Interface:* provides interfaces to external networks (such as NASCOM and MODLAN), systems (such as CMS), and devices (such as terminals and output devices)
- *Network Communications Center (NCC) Interface:* coordinates status information with the NCC
- *Operator Input:* interprets operator commands, for example in the Standard Test and Operations Language (STOL), and invokes the appropriate processes
- *Display:* formats output to the operator
- *Command:* ensures transmission of commands from CMS to the spacecraft, and monitors response
- *Telemetry:* performs engineering and scientific processing of data received from spacecraft
- *Database:* provides application-specific definitions such as telemetry fields, display formats, operator directives, etc.
- *History:* provides archiving and playback of telemetry data
- *Offline:* provides reporting and other offline functions

Table 2-3: Specific Variations Suggest Avenues for Increasing Reuse

a) Decomposition into Sybsystems Varies Across APs:

- o Some APs group the Database, History, and Offline processing into one subsystem.
- o Some APs group just History and Offline processing into one subsystem, and keep Database as a distinct subsystem.
- o External interfaces may or may not be grouped together into a single subsystem.
- o In APs of the future we may expect the Operator Input and Display subsystems to be unified into a single subsystem.

b) Some High-Level Components are Optional but Stable:

- o TAC Interface
- o MODLAN Interface
- o GMT Interface
- o Operational Data Network Interface
- o External Simulator

c) Some Requirements may be Met by a Range of Stable Options:

- o Alternative display devices
- o Alternative database management systems
- o TRDRSS *vs.* STDN satellite links

2.3 Obstacles to Reuse

We identified the following potential technical obstacles to reuse:

- o Mission-specific requirements
- o Obsolesence of host systems and external interfaces
- o Increasing complexity of system requirements

In the following subsections, we discuss how each obstacle is manifested in the systems we studied. We propose a technical approach to overcoming or avoiding each obstacle.

2.3.1 Mission-Specific Requirements

Mission-specific requirements are the most obvious obstacle to reuse. They are addressed differently in Standard Software and MAE. The Standard Software documentation identifies those software components on which mission-specific requirements are likely to have an impact. For each such component, the Standard Software documentation describes how the necessary changes could be effected (see [6], Section 7).

The developers of MAE decided to concentrate on those subsystems of an AP that are most likely to be reusable. A concept of layering was introduced to enable the selective use of MAE software. The layering concept is not presented in the MAE SRS, but some form of layering is implicit in the fact that the Telemetry, Command, and Display subsystems are not specified for MAE. These subsystems are assumed to be application-specific. We can infer additional layering from the manner in which MAE has been used by GRO and COBE. For example, COBE bypasses the MAE External Interface Subsystem in order to interface directly with various I/O devices and external systems.

MAE is an approximation to the idea of a domain model, insofar as it identifies parts that are likely to be reusable. The MAE SRS does not, however, systematically address the topic of adapting or tailoring the baseline software. Mechanisms for adaptation or tailoring may help to reduce the amount of software that is redeveloped with each new mission.

Our analysis points to one major area in which tailoring mechanisms could have a significant impact. This is the interface between the Operational Database (ODB) and the other subsystems of an AP. Command, telemetry, and display processing are largely (although not entirely) driven by parameters and definitions in the ODB. In current practice, generation of the ODB and development of the telemetry, command, and display software are separate activities. One disadvantage of this approach is the possibility of inconsistency between the ODB and the processing software. Another disadvantage is that the processing software must be redeveloped for each new application.

Recommended solution. Automated source code generation techniques can be applied to ensure consistency between the ODB and related software. A single specification in a very-high-level language (VHLL) could be used to define the ODB. As shown in Figure 2-1, the specification would be transformed into two outputs:

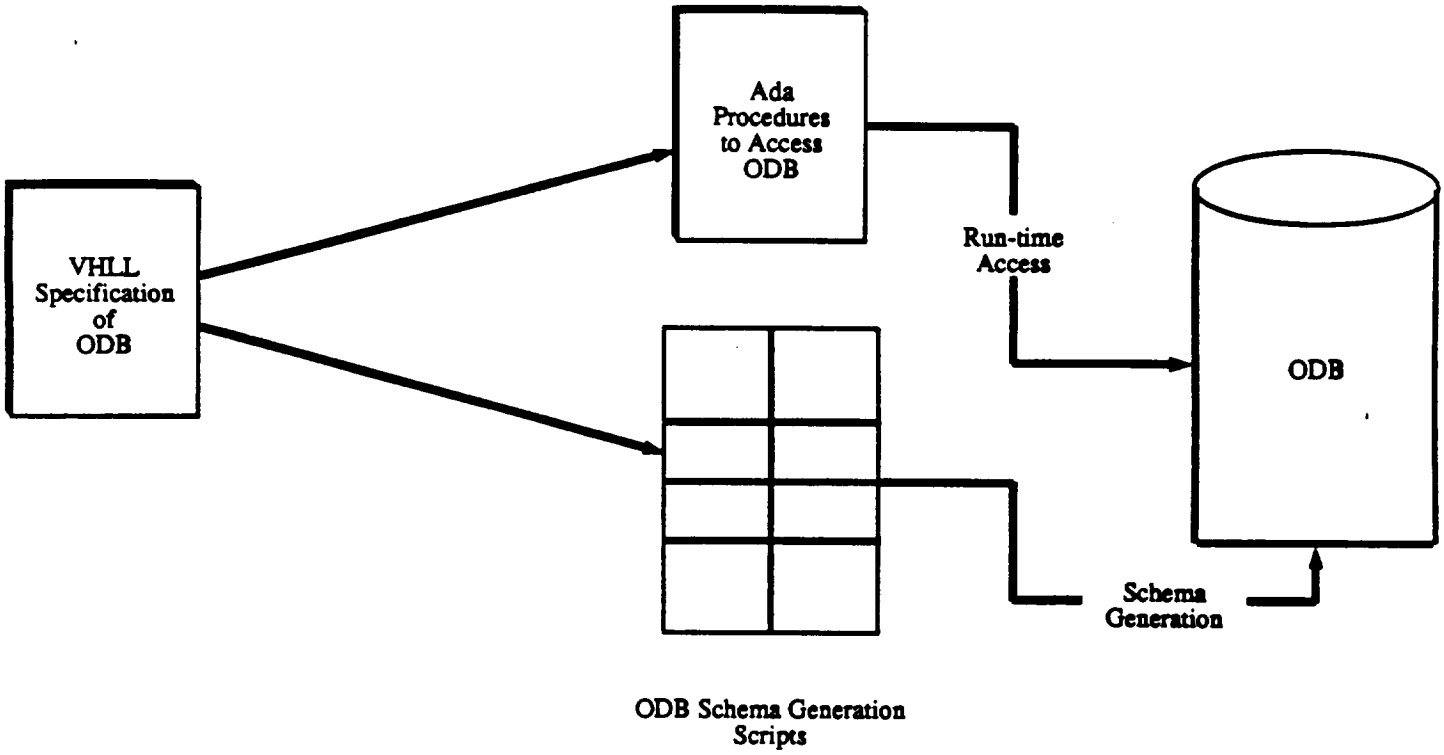

Figure 2-1: ODB Schema and Access Routine Source Code
are Generated from a Single Specification

- o Database generation scripts
- o Source code for database access routines

The database generation scripts would create the tables required by the command, telemetry, and display subsystems. The generated source code would provide access to these tables, including individual fields within each table. In current practice, these routines are manually coded for each mission. Generating the source code automatically will eliminate one major area of redevelopment.

The command, telemetry, and display subsystems would issue calls to the database access routines, which have been generated automatically. These calls would most likely take the form of generic procedure calls, i.e., the automatically generated access routines would be passed as generic parameters to any reusable command, telemetry, and display components that require database access. This approach effectively decouples the database definition from the development of reusable command, telemetry, and display components. The approach we propose is illustrated in Figure 2-2.

2.3.2 Obsolesence of Host Systems and External Interfaces

Standard Software was developed to be used in multiple missions. Five classes of potential mission-specific modifications were identified, and likely areas of modification in each subsystem of the software were documented in detail. Nevertheless, Standard Software was eventually superceded by MAE. Substantial modification occurred even in those systems that are based on Standard Software (see, for example, the DE Design Specification [2], Section One).

The development of MAE seems to have been necessitated, at least in part, by the obsolescence of the PDP-11 host of the Standard Software system. Standard Software was also superceded in the area of external interfaces. New MSOCC APs must interface with devices (e.g., display devices), networks (e.g., MODLAN), and systems (e.g., Shuttle and TDRSS) that were not present in earlier missions.

We expect that, in the near future, the device-dependent assumptions of MAE will also be superceded. MAE is hosted by Perkin-Elmer super-minicomputers, which replace the PDP-11s of previous MSOCC APs. The new hosts may remain adequate for several years. The operator interfaces of MAE, however, are based on older technology and may already be considered outdated. We predict that the use of the text-oriented STOL, with text-oriented terminal and strip-chart output, will soon be replaced by modern graphical input/output techniques using windows, menus, icons, mouses, etc., and the associated graphical monitors. Other hardware technology developments are likely to impact POCCs, especially within the planned CDOS architecture which distributes control among customer sites.

Recommended solution. The use of Ada and other standards, such as the X-Window System, may reduce some of the impact of host and device obsolesence. Object-oriented design will also help by localizing host and device dependencies. Such localization is, however, already visible in the systems we studied. The real significance of OOD is that the interfaces between components are service-based. Such interfaces permit the separation of service definitions from service implementations. This, in turn, enables the developer to change an implementation without changing the interface. *An important*

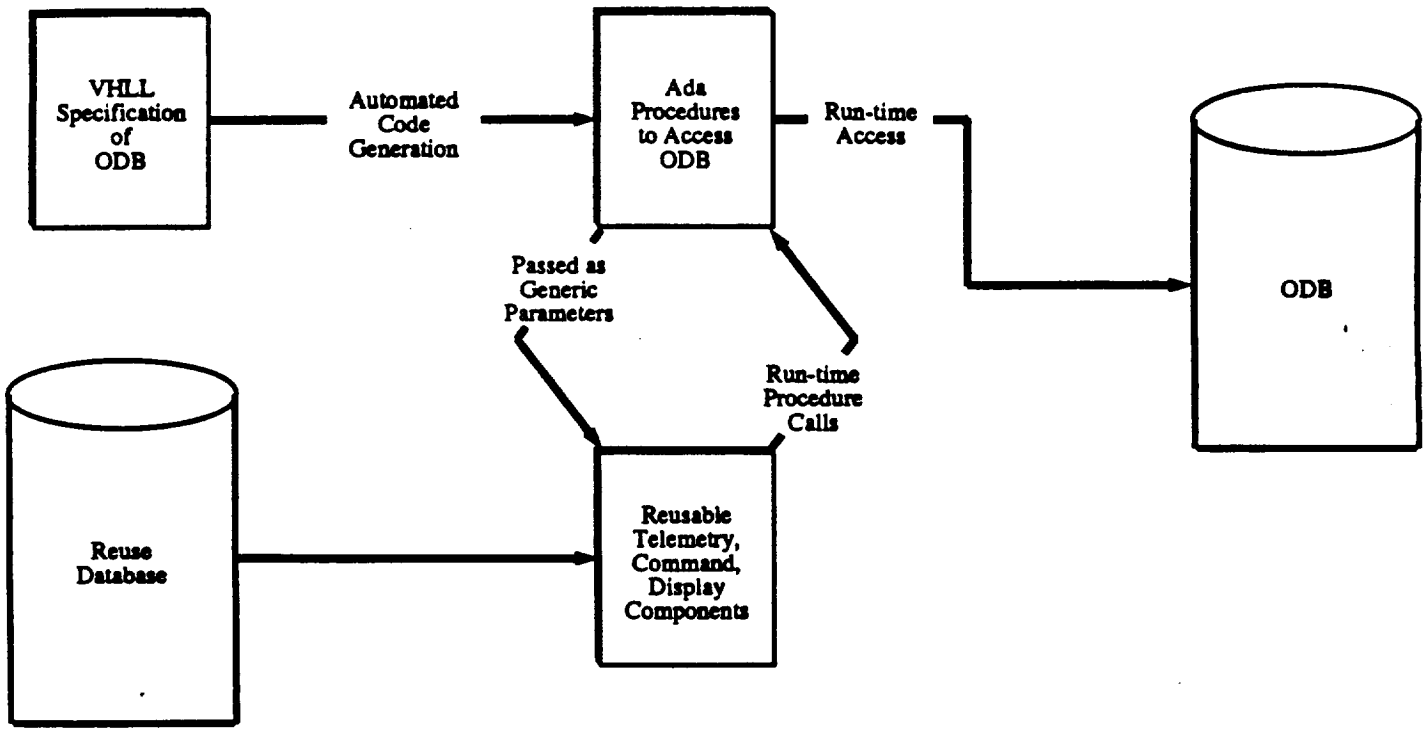

Figure 2-2: Mission-Specific Database Access Routines
may be Called by Reusable Components

part of our continued effort will be to develop object-oriented interface definitions for the standard components of a POCC AP.

2.3.3 Increasing Complexity

As functional requirements become more and more demanding, software becomes more and more complex. The addition of new functions to existing components leads, eventually, to components that are *too* complex. The response of software analysts and designers is to introduce new functional layers. By distributing functions over the increased number of layers, designers are able to reduce the level of complexity at any given layer.

We can see this pattern in the evolution of MSOCC telemetry subsystems, which is illustrated in Table 2-4. The telemetry subsystem of ISEE, whose PDS is dated 1980, contains only a few of the functions found in later APs. The complexity increases in DE, whose PDS is dated 1982. In ERBS (1984) and COBE (1986) we can see some grouping of functions that appear at the top level in DE. For example, internal simulation is handled by three top-level functions in ISEE, and by a single top-level function in ERBS and COBE. Input of telemetry in ISEE involves not only the incoming block handler but also the RSX-11 driver and the scheduler. ERBS and COBE manifest comparable complexity in the top level of their telemetry subsystems. In the GRO SRS, which also dates from 1986, we see a significantly simpler level-0 diagram for the telemetry subsystem. When we turn to the lower-level diagrams in the GRO SRS, we see many of the functions that appear in COBE at the top level (for example, extraction of minor frames, decommutation, and limit sensing).

While the level-0 diagrams for telemetry in GRO and COBE look very different, many of the same functions are performed in both systems. The problem, therefore, is to enable developers to capitalize on the commonality of functions, while using them in dissimilar designs.

Recommended solution. Evolving complexity poses an especially difficult problem for reuse because it is impossible to predict the manner in which complexity will evolve. Rather than build into the development environment an expected evolution of POCC APs, which is almost guaranteed to be wrong, we propose to provide mechanisms for *handling* evolving complexity.

The combination of object-oriented design and functional decomposition, and the graphical representations that are used to express these concepts, provide powerful methods for defining new layers of abstraction. We require techniques that facilitate *implementing* these new layers without sacrificing reuse of existing components.

We propose an approach based on graphical programming, i.e., generation of source code directly from diagrams. A combined hierarchy of Object Diagrams and Dataflow Diagrams may be used to define new layers of packages and functions. As shown in Figure 2-3, whenever a specified capability matches a component that already exists, the diagram can reference that component as an alternative to further decomposition. In this way, diagrams would be used to piece together existing components in new combinations. Source code generated from the diagrams would reference the reused components. The higher-level components developed in this manner may themselves, after suitable refinement, be added to the Reuse Database.

Table 2-4: Increasing Complexity Leads to Layering in the Evolution of Telemetry Subsystems
(Page 1 of 2)

ISEE-C Top-Level Telemetry Functions:

- o System Initializer
- o Telemetry Input Block Handler
- o TUT Generator
- o Extract Subcommutated Data
- o Attitude Output
- o Simulator Initialization

DE Top-Level Telemetry Functions:

- o System Initializer
- o Map Initializer
- o TUT Initializer
- o RSX-11 Input Driver
- o Input Block Handler
- o TUT Generator
- o Telemetry Minor Frame Extractor
- o Analog Limit Check
- o Scheduler
- o Spacecraft Configuration Monitor
- o Equation Processors
- o Limits Enable/Disable
- o Telemetry Conversion Enable/Disable
- o Dump Processing
- o Maximum/Minimum Processor
- o Simulator Initializer
- o Simulator File Setup
- o Internal Simulator

ERBS Top-Level Telemetry Functions:

- o Establish System Initialization
- o Map Initialization
- o Distribute Blocks
- o Process Telemetry
- o Write History Tape
- o Interval Data Archival
- o Quick-Look Experiments
- o Equation Processors
- o Check Analog Limits
- o Monitor Spacecraft Configuration
- o Internal Simulator

Table 2-4: Increasing Complexity Leads to Layering in the Evolution of Telemetry Subsystems
(Page 2 of 2)

COBE Top-Level Telemetry Functions:

- o Configure Telemetry System
- o Route NASCOM Blocks
- o Strip Minor Frames
- o Decommutate Frames
- o Limit Check Data
- o Process Equations
- o Monitor Spacecraft Configuration
- o Accumulate Interval Archival Data
- o Extract Quick-Look Data
- o Fabricate Telemetry Block Internally

GRO Top-Level Telemetry Functions:

- o Process NASCOM Telemetry Blocks
- o Process Real-Time Telemetry
- o Process OBC Dumps
- o Process Periodic Telemetry Functions
- o Process CAPS

GRO Process Real-Time Telemetry Subfunctions (Level 1):

- o Extract Minor Frames
- o Decommutate Telemetry Parameters
- o Perform Decommutation Special Processing
- o Perform Limit Sensing
- o Perform Delta Limit Sensing

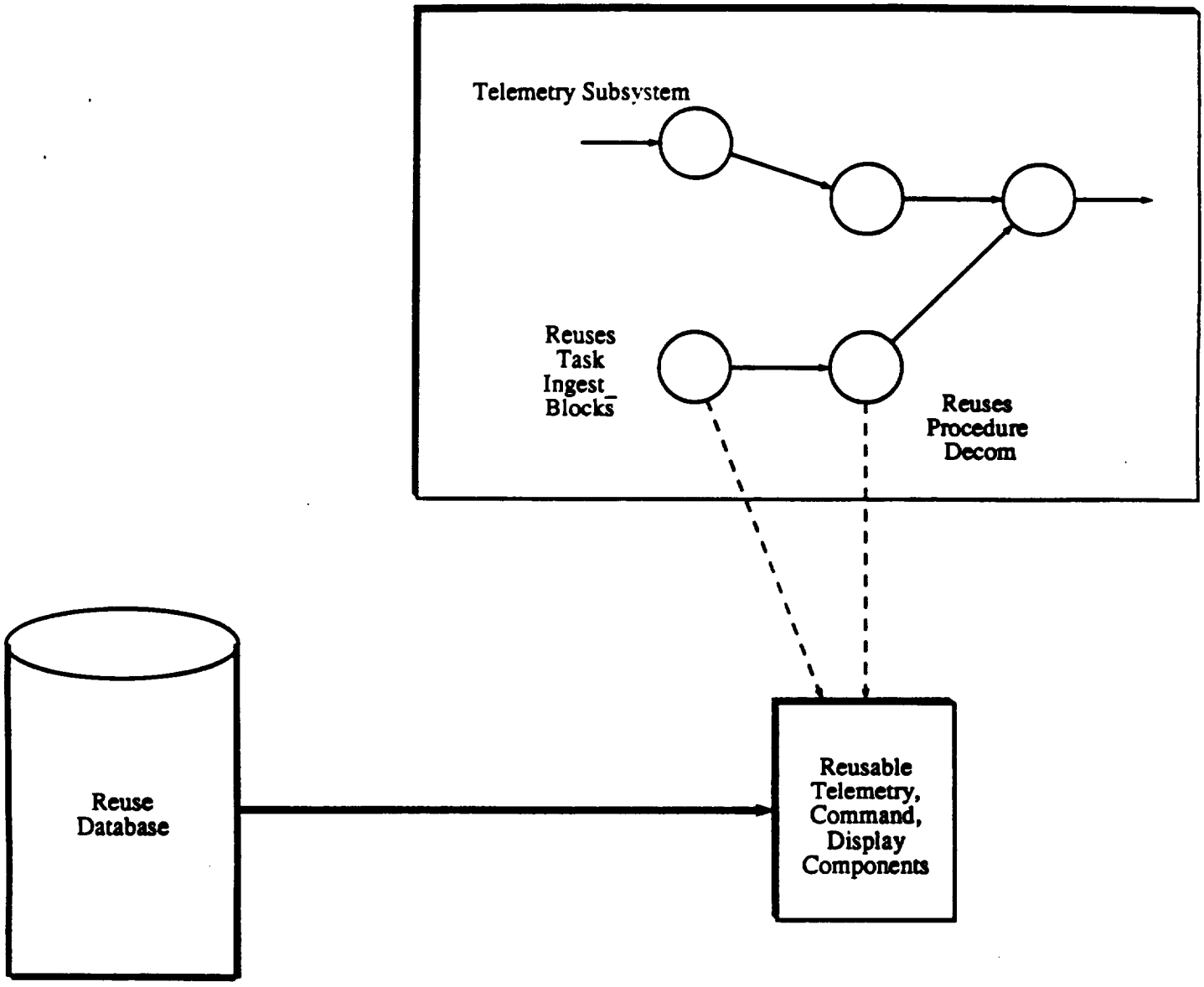

Figure 2-3: Object and Dataflow Diagrams may Reference Reusable Components

This approach would be especially suitable for handling evolving complexity in telemetry processing. In the systems we studied, the dataflow diagrams for telemetry reveal an overall paradigm of *pipeline processing*. As the telemetry stream is read in, each element is passed to a series of maps, filters, state machines, etc., with possible splitting and merging of streams at any point in the pipeline. Under the Department of Defence STARS program, CTA has already developed a tool that takes, as input, a description of such a pipeline together with the functions to be performed at each step, and generates, as output, Ada source code that implements the pipeline [14]. This is an example of the kind of generative technology that could be employed.

3 TECHNICAL APPROACH TO SEMI-AUTOMATIC DEVELOPMENT

The reusability analysis in Section Two suggests that we can effectively combine four techniques for semi-automatic development of POCC software:

- o Dialog-based specification
- o Navigation of the Reuse Database
- o Domain-specific Very-High-Level Language (VHLL)
- o Graphical programming

3.1 Dialog-based Specification

Some of the high-level differences between APs were identified in Table 2-3a. These variations do not appear to offer any major problems for reuse: they are few in number and easily defined, and can be presented simply as alternatives to the developer.

In the scenario we envision, the developer would begin by choosing a top-level architecture for the system. The developer knows that a POCC AP contains certain standard subsystems, but the exact arrangement of these subsystems is open for analysis since in the past different missions have done things differently. The developer wants to know what options have been chosen in the past, and what rationales led to those decisions. He can then compare the rationales with the current requirements, and make a decision accordingly.

We envision the environment presenting the developer with questions that represent the alternatives. For example:

Functions:

NCC Interface Required?
Command Verification Performed?
External Simulation Performed?
Expert Systems Required?

Levelling:

Combined External Interface Subsystem?
Combined Operator I/O Subsystem?
Combined Database, History, and Offline Processing?

Placement:
Initialization performed where?
Database(s) distributed throughout other subsystems?
Expert Systems distributed over Operator Positions?

If the developer needs clarification of any of these questions, the environment can present the alternatives that are available in the Reuse Database. Once the dialog is

complete, the environment would retrieve the necessary subsystems and components and would synthesize the top-levels of the AP

As long as the options are relatively few, the environment need not have an explicit rule-base in order to conduct and interpret the dialog. Greater flexibility could, however, be achieved by providing a rule-base and inference engine. Such rules could, for example, determine how various options are interrelated. They might also determine restrictions or necessary steps to be taken when integrating subsystems under various conditions.

In the absence of a rule base, the rules for integrating subsystems could be deduced from the relationships present in the Reuse Database, as illustrated in Figure 3-1. Object-oriented interfaces are essential for this approach to work. The data flow model, which was used in the majority of the SRSs that we examined, does not provide sufficient information to enable a developer even to consider reuse at the subsystem level. For example, the data flow model does not describe the conditions under which data flows between subsystems occur, nor does it describe the control relationships between subsystems. The decision to reuse a subsystem must, therefore, be based upon an analysis of lower-level documentation, e.g., lower-level dataflow diagrams, and interface control documents. An object-oriented definition of subsystem interfaces would alleviate this problem by providing the necessary information at the highest level of documentation. The information could initially be provided in the form of Object Diagrams. The relationships documented in these diagrams ("uses," "provides," etc.) would then be incorporated in the Reuse Database. The information required for reuse of a subsystem would then be available to the environment's dialog interpreter.

3.2 Navigation of the Reuse Database

Having chosen a top-level architecture, the developer proceeds to decompose each subsystem. He locates each subsystem as a category (i.e., keyword) in the Reuse Database. Starting from each subsystem name, the developer locates the possible components of the subsystem by navigating down the relationship "contains."

For stable components, the Reuse Database provides a convenient form in which to represent the range of options. The developer can navigate over the available options and select those that are needed. For example, a developer who is familiar with STOL might learn about the existence of alternative operator interfaces approaches by navigating through the options. Table 3-1 illustrates some of the components that the Reuse Database would contain.

3.3 Domain-Specific Very-High-Level Language

Many of the mission-specific properties of an AP are reflected in the schema of the AP database. As described in Section 2.3.1, a significant amount of redevelopment can be avoided by automatically generating database interface source code at the same time that the schema is generated.

Transformation of domain-specific VHLL specifications into ordinary source code is a reasonably mature discipline. The approach we recommend is based on the "pre-spec" method developed by CTA for NASA under another program. A pre-spec is a program, written in Ada, that is used to specify a systen at a very high level of abstraction. Ada is, in effect, used as the VHLL. Compilation of the pre-spec results *not*

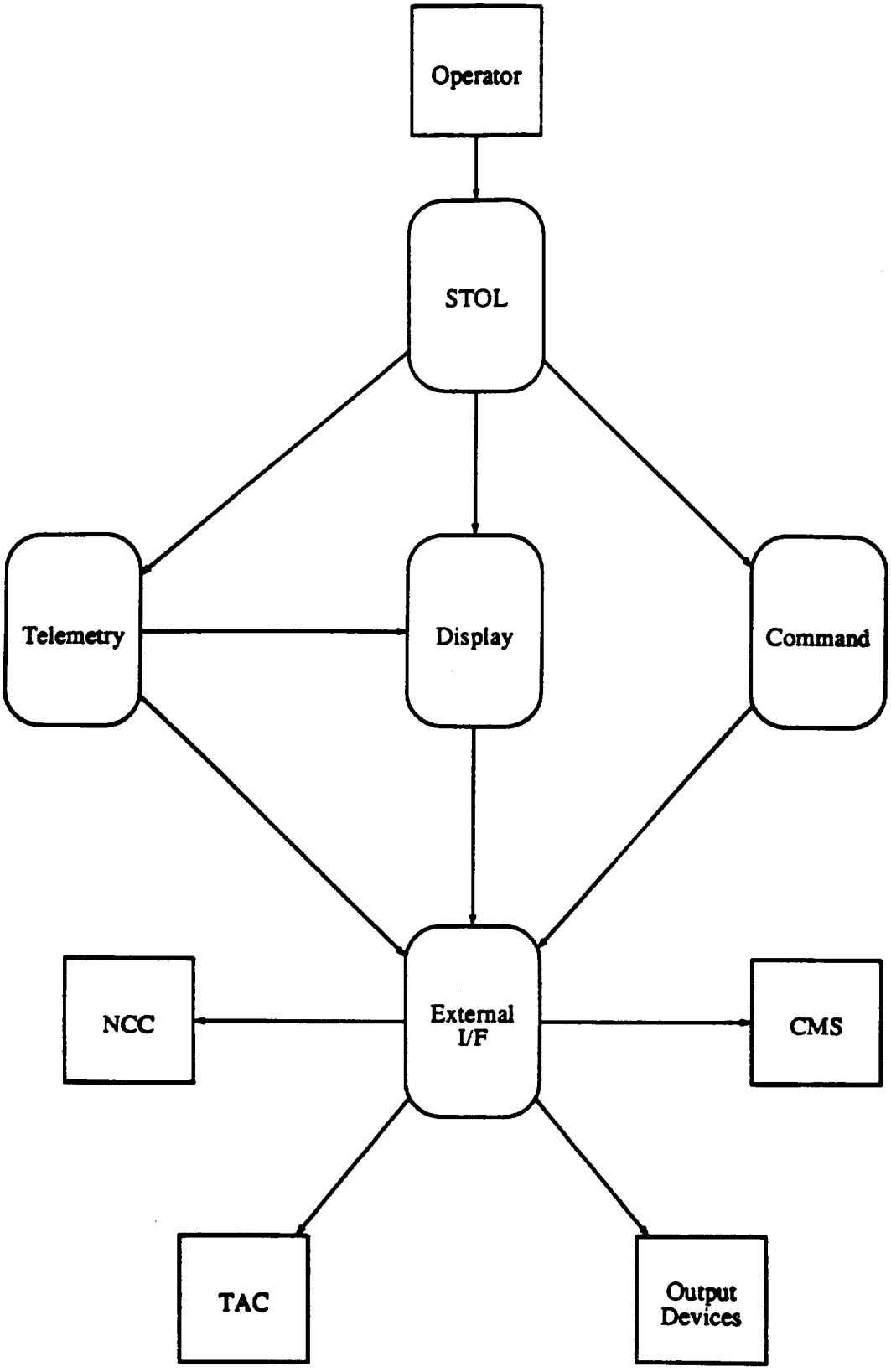

Figure 3-1: Relationships in the Reuse Database
Determine how Components should be Integrated

Table 3-1: The Reuse Database will Contain Common Components of POCC APs

- *POCC Subsystems:*

  External interface
  Operator
  Telemetry
  Command
  Database

- *External interfaces*

  MODLAN
  ODN
  DOCS
  FDF
  CMS
  TAC

- *Operator interfaces*

  STOL
  Terminal
  Strip Chart Recorder
  Graphical display

- *Telemetry functions*

  Strip minor frames
  Decommutate telemetry data
  Extract data (various types)
  Limit sensing
  Dump Image maintenance
  Quick-look processing
  Equation processing
  Monitor spacecraft configuration
  Internal simulation
  Interval data archival

- *Command functions*

  Transmit command
  Verify receipt

in the desired program, but rather in a code generator. Execution of this code generator results in the desired program.

As shown in Figure 3-2, this method works by hiding the details of code generation in a support package, which is brought in when the pre-spec is compiled. The support package serves as the definition of the VHLL. The principal advantage of the pre-spec method over other VHLL methodologies is that domain-specific languages can be created and refined, without introducing an ever expanding set of language constructs. By using Ada as the VHLL we provide the flexibility to tailor the VHLL to an application, while retaining a familiar syntax. The pre-spec method has been applied in a variety of domains, including the database domain which is where we propose to apply it.

3.4 Graphical Programming

In Section 2.3.3 we identified the problem of evolving complexity, and we proposed graphical programming as a means of handling this problem. Graphical programming would provide a simple means of combining reusable components into new, higher-level components. We propose to generate Ada source code from a combination of Object Diagrams and and variant of Dataflow Diagrams. Object Diagrams would be used to combine or integrate reusable packages. Dataflow Diagrams would be used to integrate reusable functions and procedures.

CTA has developed, under its internal research and development program, a code generation tool that converts a hierarchy of Object Diagrams into Ada package specifications and skeletal package bodies. This program is now addressing the automatic generation of procedure bodies from a variant of Dataflow Diagrams, which we call Functional Diagrams. Functional Diagrams are a graphical idiom for expressing processing logic without side effects. The side-effect-free programming discipline results in code that is more easily demonstrated to be correct, and that is easier to maintain than code that contains side effects. The tool now being developed by CTA could be used in a demonstration of the overall approach to semi-automatic development of POCC software.

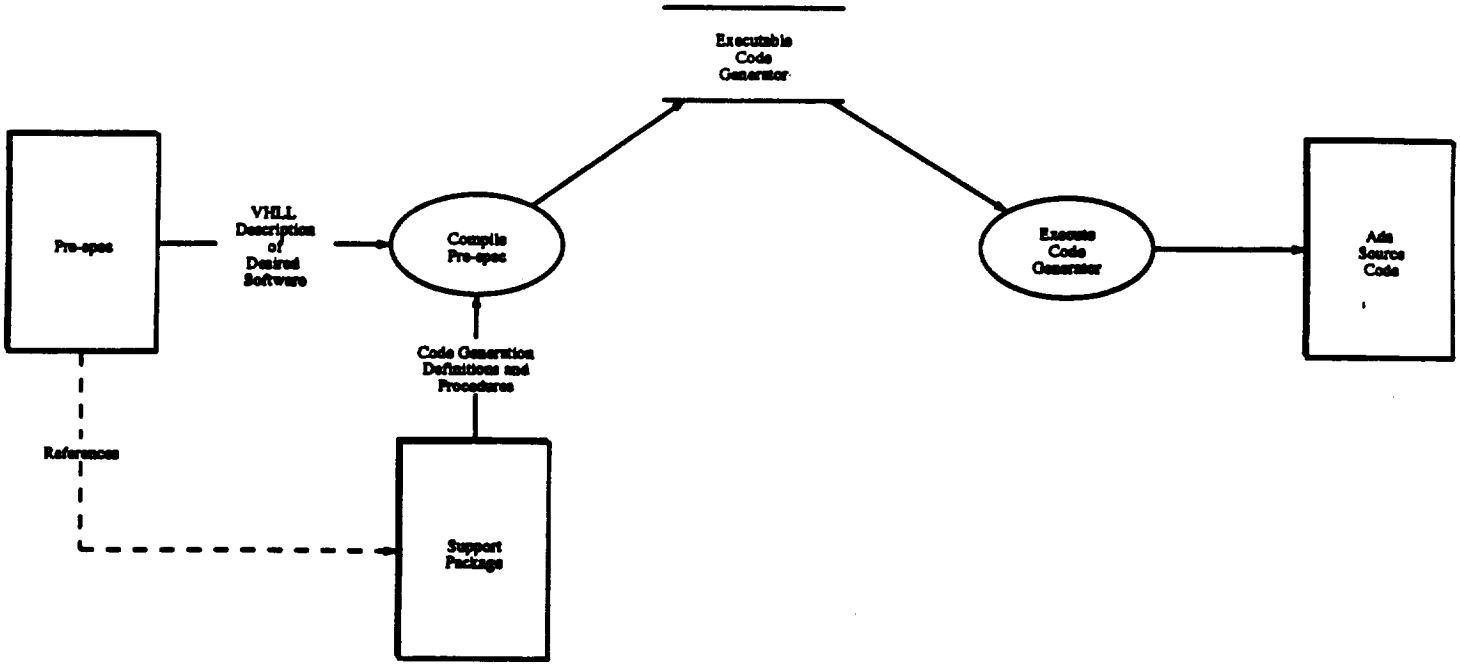

Figure 3-2: Pre-Spec Semantics are Determined by a Pre-Spec Language Support Package

4 THE NEXT PHASE: DEMONSTRATING SEMI-AUTOMATIC DEVELOPMENT

We have identified the principal technical obtacles to reuse of POCC AP software, and we have described four development technologies that can help to overcome these problems. We recommend synthesizing a prototype software development environment that demonstrates how these technologies would work together. Figure 4-1 illustrates the integrated concept.

The purpose of such an effort would be to demonstrate the capability to synthesize a POCC AP rapidly. One aspect of the effort would consist of enhancing the current Software Reuse Environment [4] with tools that support dialog-based specification, the domain-specific VHLL, and graphical programming. Another part of the effort, which is at least as important, would consist of developing object-oriented specifications for reusable POCC AP components. The latter process might be called "domain synthesis." It is the natural continuation of our study of reusability in the POCC AP domain.

The objective of the next phase is rapid synthesis or "semi-automatic development" of POCC APs. Significant human intelligence will still be required in the development process, i.e., in the database definition process, in the graphical programming process, and probably in tailoring reusable components to application-specific requirements. The goal is not so much to automate as to raise the level of abstraction at which developers must think. The effort is analogous, in this respect, to the development of compilers for high-level languages such as FORTRAN and Ada--with the significant difference that we focus on a single application domain. This focus will enable us to take advantage of specification, design, and implementation conventions that cannot be assumed in the general case.

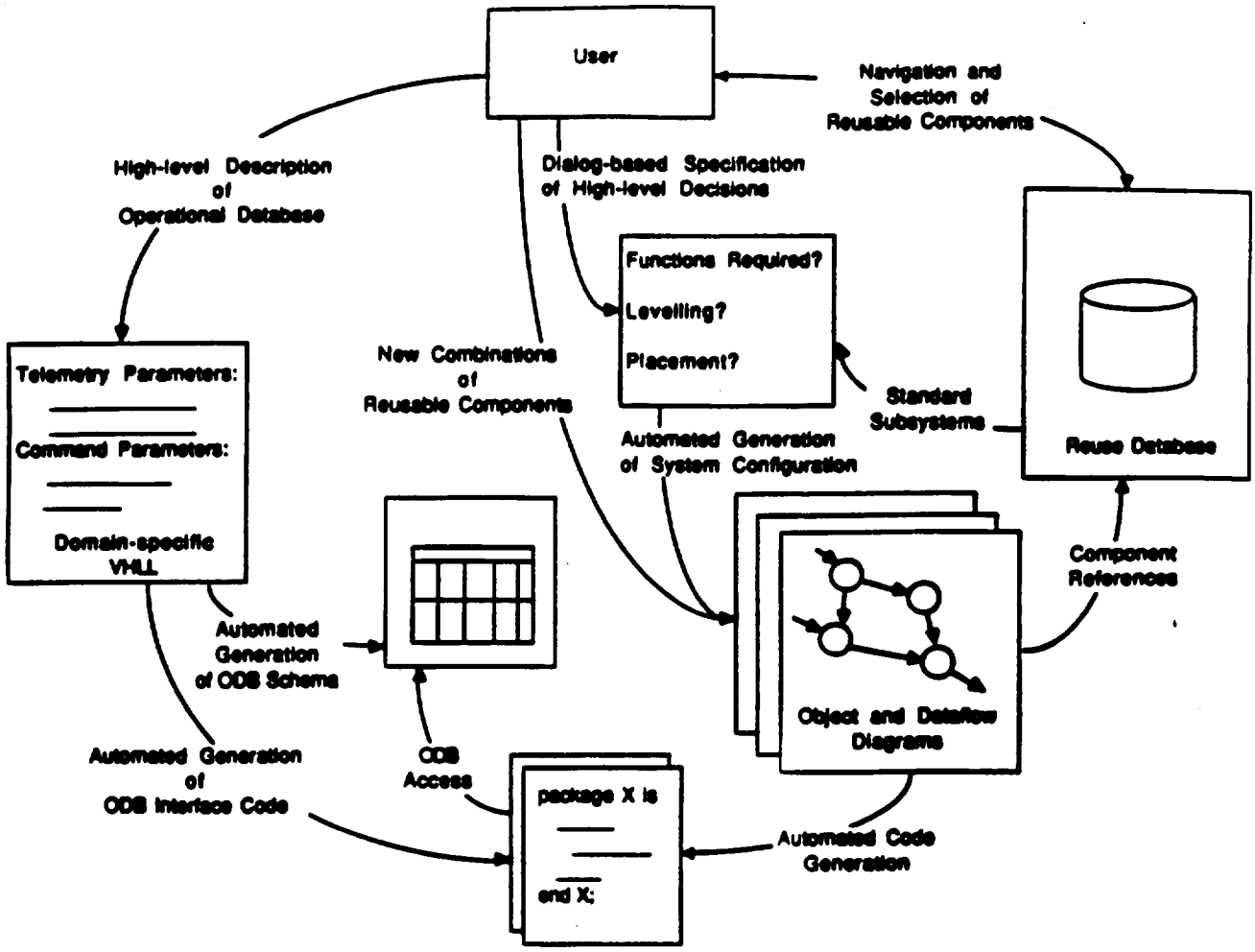

Figure 4-1: Four Automation Techniques Combine to Support
POCC Software Development

REFERENCES

[1] Bailin, S., Dillencourt, M, Heyliger, G. *Automation in Software Development Environments*. Prepared for NASA Goddard Space Flight Center. Computer Technology Associates, Inc. May 16, 1986.

[2] Bailin, S. *Scenarios of Software Development in Code 500*. Prepared for NASA Goddard Space Flight Center. Computer Technology Associates, Inc. January 12, 1987.

[3] Bailin, S., Moore, J., Rogerson, A. *Towards Automated Software Development in Code 500: Short and Long Term Recommendations*. Prepared for NASA Goddard Space Flight Center. Computer Technology Associates, Inc. January 30, 1987.

[4] Bailin, S. *Software Reuse Environment User's Manual*. Prepared for NASA Goddard Space Flight Center. Computer Technology Associates, Inc. March 30, 1988.

[5] Bailin, S. C. and Moore, J. M. *An Operational Concept of Software Reuse*. Prepared for NASA Goddard Space Flight Center. Computer Technology Associates, Inc., June 15, 1987.

[6] *Control Center Standard Software Final Report*. Westinghouse Electric Corporation report to NASA GSFC. July 27, 1979.

[7] *Program Design Specifications for International Sun-Earth Explorer-C*. OAO Corporation for Westinghouse Electric Corporation, specification to NASA GSFC. April 1980. (Updated by Bendix Field Engineering Corporation for NASA GSFC. May 1987.)

[8] *Dynamic Explorer Control Center Software Manual*. Sperry Univac specification to NASA GSFC. January 29, 1982.

[9] *Earth Radiation Budget Satellite (ERBS) Software Requirements Analysis*. Computer Sciences Corporation specification to NASA GSFC. Updated September 30, 1983.

[10] *MSOCC Applications Executive (MAE) Software Requirements Specification*. Sperry Corporation specification to NASA GSFC. November, 1985.

[11] *Cosmic Background Explorer (COBE) Payload Operations Control Center (POCC) Applications Processor (AP) System Requirements Specification*. BFEC specification to NASA GSFC. October 1986.

[12] *Gamma Ray Observatory (GRO) Payload Operations Control Center (POCC) Applications Processor (AP) Software Requirements Specification*. Sperry Corporation specification for BFEC report to NASA GSFC. February, 1988.

[13] *Domain Analysis for Control Center Software*. CTA INCORPORATED for Computer Sciences Corporation, report to NASA GSFC. September 2, 1988.

[14] *Ada Streams Package Users Manual.* CTA INCORPORATED report to Naval Oceans Systems Center. April 30, 1988.