DATA PROBE: A TOOL FOR EXPRESS-BASED DATA

KC Morris

Factory Automation Systems Division
National Institute of Standards and Technology
Gaithersburg, Maryland

ABSTRACT

The problem of sharing data has many facets. The need to share data across multiple enterprises, different hardware platforms, different data storage paradigms and systems, and a variety of network architectures is growing. The emerging Standard for The Exchange of Product Model Data (STEP), being developed in the International Organization for Standardization (ISO), addresses this need by providing information models, called *application protocols*, which clearly and unambiguously describe data. The validity of these information models is essential for success in sharing data in a highly automated engineering environment.

This paper describes the Data Probe: a tool for examining, editing, and managing EXPRESS-based data. The Data Probe tool supports the validation of STEP application protocols. The paper includes a description of the software architecture, the initial implementation, and plans for future enhancements. The software is designed as independent components which can be incorporated into other STEP-related systems or software requiring general purpose editing tools for structured information.

The initial version of the Data Probe tool is based on two implementation mechanisms defined within STEP: the conceptual modeling language EXPRESS and the STEP exchange file format. Future work will focus on integrating a database system into the software. The software architecture and the use of object-oriented techniques enables code reusability and system extensibility and has been instrumental for a phased implementation.

The software is under development at the National Institute of Standards and Technology and is in the public-domain. The software supports the Validation Testing System, part of the Application Protocol Development Environment, at the CALS-sponsored National PDES Testbed. (PDES, Product Data Exchange using STEP, is the U.S. effort in support of the international standard.)

INTRODUCTION

The emerging Standard for the Exchange of Product Model Data, a project of the International Organization for Standardization and commonly referred to as STEP[1], addresses the need to share data in a complex computer environment. STEP will provide a basis for a common understanding and communication of data, thus allowing it to be shared. STEP includes definitions of information models and mechanisms for representing the models and related data. The information models communicate the structure and the semantics of the data necessary to inter-operate between different computer systems. The validity of these information models is essential for success in sharing data in a highly automated business environment — the models must be shown to be both useful and usable for their intended purposes. Demonstration that the information models support the needs of applications is the most direct method of validating these models (Mitchell, 1991).

The information models within STEP are integrated and organized into *application protocols*. An application protocol addresses a specific application area and contains an information model written in the conceptual modeling language EXPRESS (ISO, 1992b). The information model describes the data in the application area. The *Data Probe* tool described in this paper is being developed to support validation of these application models[2]. The Data Probe is a tool for editing data and is under development at the National PDES Testbed[3].

[1]The Standard for the Exchange of Product Model Data (STEP) is a project of the International Organization for Standardization (ISO) Technical Committee on Industrial Automation Systems (TC184) Subcommittee on Industrial Data and Global Manufacturing Programming Languages (SC4). For an overview of the standard refer to *Part 1: Overview and Fundamental Principles* (ISO, 1992a).

The background of validation software at the National PDES Testbed is presented in the following section. Subsequent sections describe and discuss the design of the VTS software architecture and the Data Probe tool. Further details of the methodology used for validation and requirements for automating that process are described in other documents (Mitchell, 1991)(Morris, 1991a)(Morris, 1991b)(Mitchell, 1992).

This paper is directed at software designers, developers, and project managers. Developers of tools relating to STEP and, in particular, to the validation of application protocols will be interested in the architecture and the approach taken to software development. This architecture is also relevant to similar software projects which involve the presentation and manipulation of structured data. An object-oriented approach is used for the architecture, design, and implementation of the Data Probe software.

BACKGROUND

This section provides a brief background of the needs for validation testing software for the reader unfamiliar with the validation process for application models. For further information on these topics the reader is referred to (Mitchell, 1991) (Morris, 1991b) (Mitchell, 1992). An overview of the need for validation software is presented here. A discussion follows of prior experience with software used for the validation of application models at the National PDES Testbed.

The Need for Validation Testing Software

People have many means of communicating ideas and information. When someone writes a paper to convey a point, readers of the paper can judge the position presented and determine whether or not it is valid. The validity of the point raised in the paper is not judged by the tools which were used to prepare the paper. The writer could use pencil and paper or a sophisticated word processing system to prepare the manuscript — the tools used would not matter to the reader in judging the validity of the point raised. The validity of the paper is judged on the evidence provided to supports the point that the paper raises.

On the other hand, a means of sharing information between computer systems must be very rigidly defined, hence a standard emerges. Such a standard can be specified as an *information model* which captures the meaning of the information in the context of a particular usage, or *application area*. The validity of such a standard is based on the usefulness of the content *and* the computability of the presentation format. In this case the tools used to prepare the standard and to produce the evidence that it is capable of supporting its intended usage are of the utmost importance. If the proposed standard is not demonstrated to support the computerized sharing of data, it will never be used. Furthermore, due to the complexity of the standard, tools are necessary to assist people in analyzing the completeness and accuracy of the specification.

Two aspects are important for validating such a standard:

- the meaning of the standard must be clear and complete, and
- the standard must be in a format which allows for the computerized sharing of data.

These two aspects combine to effectively communicate and successfully implement the ideas presented in the standard. Judging the first aspect — the *content* of the standard — is much like the judgement the reader of a paper makes as to its quality. However, the second aspect — the *computability* of the standard — is only tested by representing the information in a computer and demonstrating that the representation is capable of meeting the data access needs for the applications involved in the sharing. The computerized implementation of the standard information model also helps the person who must judge the content to analyze the clarity, accuracy, and completeness of the model. This assistance is necessary for a person to be able to reliably evaluate complex information models.

The software used to validate application models at the National PDES Testbed serves two purposes:

- to assist users in tracking and manipulating the vast amount of complex information involved in validating the content of the application models, and
- to demonstrate that the models can meet the needs of the computer applications which they are intended to support.

Application protocols are validated by simulating the data access needs for the particular application area (Mitchell, 1991) which they are intended to support. This strategy addresses both aspects necessary for validating an application model. It provides a means for people to judge the content of a model, and at the same time it demonstrates the usability of the model on a computer.

Assistance in managing the information in the model and the additional information involved in validating the model is necessary due to the complexity of this information. The application of software tools to assist in these tasks greatly increases the productivity of the people validating the models. Furthermore, the quality of the tools can also impact the testing process. For instance, the project schedule for the validation of one of the early application

[2]The term *application model* is used throughout this paper to refer to the domain specific information model which is being evaluated. The application model may be part of an application protocol or any other information model such as an application resource model (also in STEP) or a similar model which is not included in the international standard.

[3]The National PDES Testbed is located at the National Institute of Standards and Technology. Funding for the work described in this paper has been provided by the Department of Defense's Computer-Aided Acquisition and Logistic Support (CALS) Office. The work described is funded by the United States Government and is not subject to copyright.

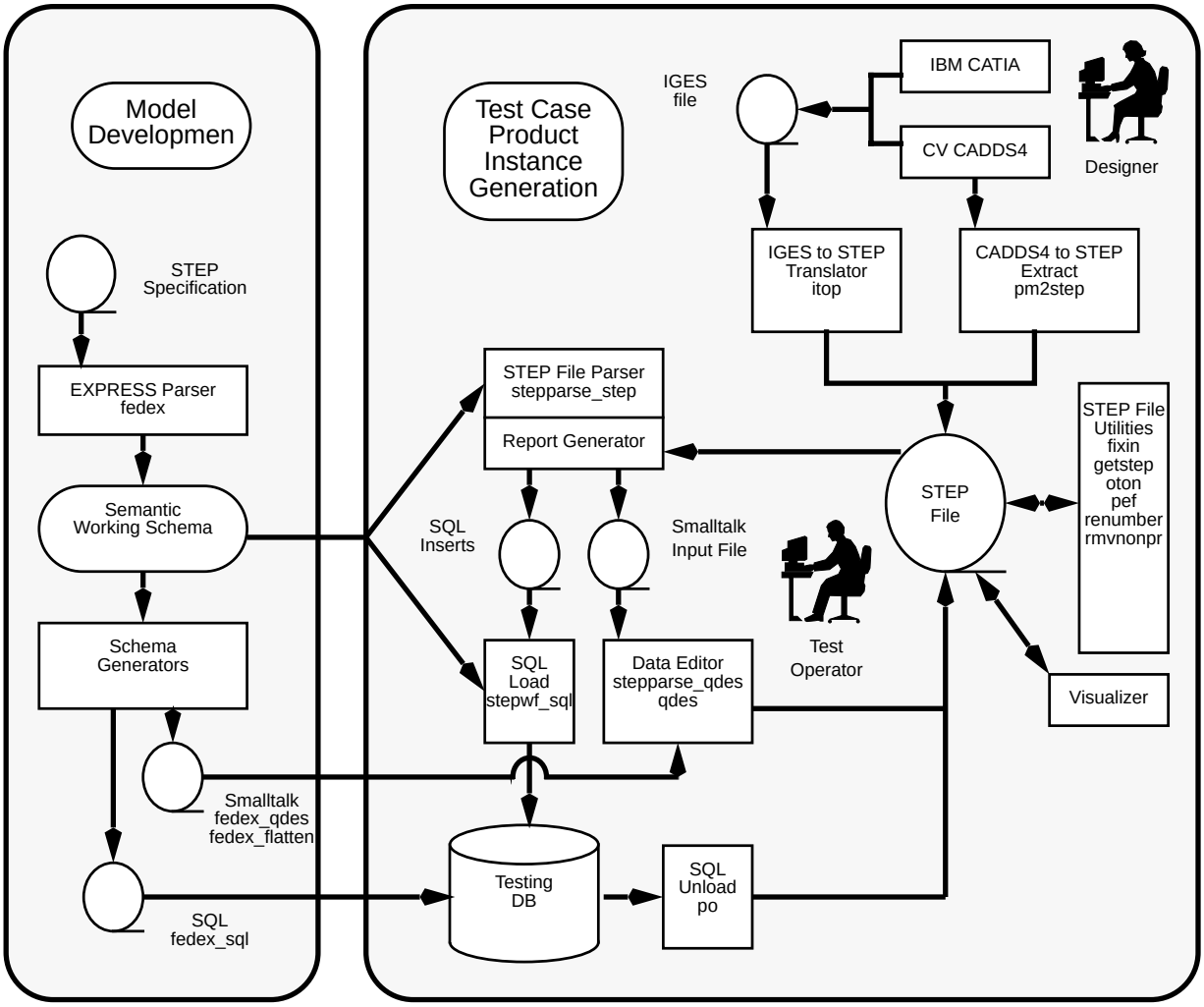

FIGURE 1 : INTERIM SOFTWARE AT THE NATIONAL PDES TESTBED

models for configuration controlled design data was severely impacted by the lack of reliable and efficient tools (PDES, 1990a).

In addition, the content of an application model can be more reliably judged with the aid of software. Initial rounds of validation testing uncovered significant flaws in information models which had been proposed for standardization. One specific example of such flaws was found in the Geometry model in STEP: the proposed model did not require objects to be founded in geometric space. This flaw created ambiguity in the meaning of data exchanged according to that model. By simulating the application needs, testers were able to analyze the model and uncover this flaw — after it had gone undetected by visual inspection of the source text for over two years.

Validation Testing Software at the National PDES Testbed

The software to support validation simulates the data access requirements of an application area[4]. As stated earlier, this strategy addresses both aspects necessary for validating an application model — it supports validation of the content of a model, and it demonstrates the computability of the model. The simulation reflects the intended usage of the application model and is essential

[4]Other software requirements for validation (Morris, 1991a) include support for clerical aspects of the process, such as preparation of documentation. These aspects are not addressed in the architecture. These requirements can be supported using commercially available software, such as word processing systems.

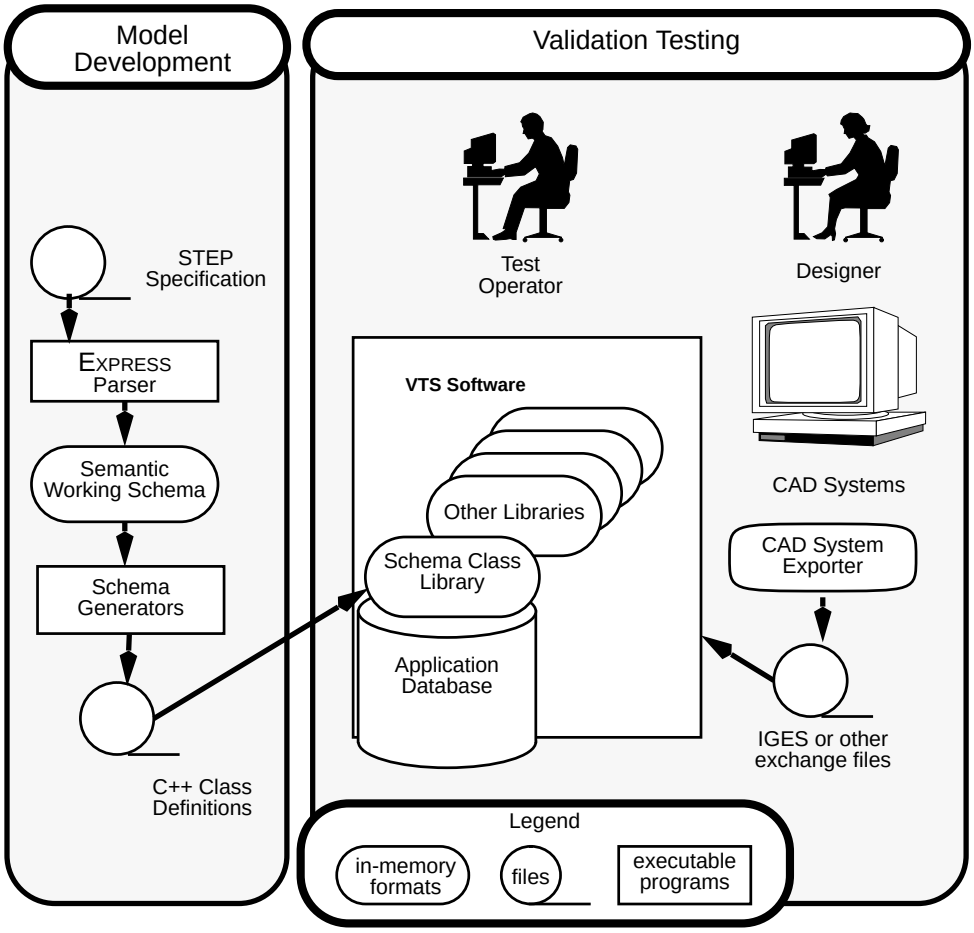

FIGURE 2 : THE VALIDATION TESTING SYSTEM SOFTWARE ENVIRONMENT

for validating the model. The method allows people to fully analyze the content of a model against requirements for an application area. Furthermore, the testing process is not verifiable if the tests are not computer processible and repeatable.

In addition, the software assists in managing the complexity of the information involved in the testing process. This information includes

- product data associated with an application model,
- relationships within the data,
- the application model as defined in EXPRESS, and
- the english description of the application model.

The Data Probe tool assists in preparation of product data to be used in testing an application model. The software supports the editing, browsing, and formatting of product data, and browsing of the structure of the application model. Planned future enhancements include support for the execution of the actual tests.

The National PDES Testbed has been used for STEP validation testing since 1989. The early software which supported the testing process, the *interim system***,** consisted of a set of tools which were not integrated.**FIGURE 1** taken from the *Validation Testing Laboratory User's Guide* (Breese, 1991), illustrates the problems of the interim system. The tools in this system were collected from a variety of sources, and, as a result, they operated on a variety of hardware platforms and in a variety of software environments. In the interim system the user needed to be familiar with a number of executable programs and their interfaces. In addition, the method of sharing data throughout the testing process was through exchange of data files between the different tools. The data was represented in memory in one of a number of different types of data structures depending on the tool.

The Data Probe integrates the functions supported by the set of tools in the interim system. The integration of the functions more

efficiently automates the testing process. The Data Probe software improves on the existing system in the following areas:

- the number of errors is reduced by reducing the frequency of data translation and human intervention;
- the amount of time needed for the testing process is reduced by improved performance and automation of the workflow;
- the time needed for learning to use the software is reduced by providing a single user interface to the system;
- inconsistencies in the data are reduced by providing more sophisticated support for data editing and creation; and
- the potential for errors is reduced through better and more extensive error checking.

The interim system required data translation every time data was moved between activities in the testing process. The translation process introduced errors or inconsistencies, and the associated manual steps, such as importing and exporting the data, were time consuming. With respect to data editing, the interim system did not track which portions of the data were complete and which needed further development. Manual support for this function was extremely difficult, given the amount of data, and lead to inconsistencies in the data. In addition, the manual configuration of the different versions of all the intermediate data files was error prone. The Data Probe software, illustrated in **FIGURE 2**, allows users to operate through a single interface, rather than with each tool separately.

Planned future enhancements to the Data Probe include the use of a database system, rather than exchange files, as the primary means of data sharing[5]. In the interim system, data was assembled from the various tools and manually integrated using exchange files to create a single data set. This manual process forced the testing process to revolve around the availability of data and tools to provide the data, rather than the needs for testing particular aspects of the application model. The use of a database system which can be shared by the different tools will make the process smoother, thereby allowing the user to concentrate on the validation activities.

The remainder of this paper describes the software which supports the Data Probe tool. The Data Probe tool is based on the Validation Testing System (VTS) software. The VTS software is composed of reusable software libraries which support the different functional areas for automating the validation process. These libraries are combined to create a Data Probe tool which supports the validation of a specific application model.

[5]Note that the database environment will not preclude the use of exchange files as a means of importing and exporting data into and out of the system. In particular data files are used to import data from external CAD systems. Exchange files are also used for sharing data until a database has been integrated into the system.

The architecture is designed to isolate changes to the system. This allows the software to be extended with additional functionality as available and as new requirements emerge. For example, the initial implementation has focused on the implementation of a data editor. The next aspect to be addressed is the integration of a database system into the software. The integration of the database system should be transparent to the users of the data editor; however, the integration will enable the expansion of the functions supported by the tool.

THE VTS SOFTWARE ARCHITECTURE

The VTS software architecture integrates modular software libraries which are incorporated into a single system to provide the functions needed for the validation process. The use of object-oriented techniques and standard interfaces enables software reusability. The system uses as much software as is available from external sources. When such software is unavailable, the necessary software has been developed. Specifically, support for the implementation methods specific to STEP were developed.

Since STEP is a developing standard, the mechanisms for its implementation have not been stable. Furthermore, these mechanisms have been developed concurrently with the application models which the VTS software is used to validate. These mechanisms include the EXPRESS language in which the application models are specified and the data interface formats, such as the exchange file format (ISO, 1992c) and the STEP Standard Data Access Interface (SDAI) (ISO, 1992d). The software at National PDES Testbed must be quickly and easily adaptable to new versions of the mechanisms.

The following goals have influenced the design of the VTS software:

- to minimize the need for data translation by providing an integrated system which supports a broad range of functions,
- to provide a single end-user program,
- to easily transition the software to support a new application model,
- to enable different style user interfaces to be developed,
- to allow for the integration of externally developed software into the system, and
- to develop reusable software.

The previous section described the high-level design of the system — an integrated software system with a single user interface. This design is reflected in the first two goals listed above. The remaining goals on the list above have influenced the modular organization of the VTS software and are discussed throughout this section.

Establishing the Testing Environment

An application model is represented in many formats throughout the validation process: English, EXPRESS and EXPRESS-G (ISO, 1992b), other graphical modeling formats, one or more programming languages, and computer memory formats. From the perspective of the end user the interface to the VTS software is through the EXPRESS description of the application model; however, from the perspective of the software developer the programming language format is of primary importance (Clark, 1990). In the interim system many different programming formats are used by the various tools; however, in the VTS software a single format is used: C++[6] (Stroustrup, 1990). The C++ representation of the application model can be automatically generated from an EXPRESS description.

FIGURE 3 illustrates the VTS software development methodology for producing a testing environment for a specific application model. An application model, as represented in a library of data structures and access functions, is integrated into the VTS software. The application model, described in EXPRESS, is translated into a software library (the upper-left side of **FIGURE 3**.) This library is then installed in the VTS software to generate a new testing environment for that model. (See the description of the Data Probe tool in the following section.) This library changes for each application model being tested. In order to minimize the difficulty in the transition to a new testing environment it is important to be able to automatically generate these libraries.

The pieces above the center line in **FIGURE 3** reflect system requirements, or inputs into the software development process; the pieces below the line illustrate the structure of the VTS software for a runtime system. The representation of the application model used by the software results from the direct translation of an EXPRESS schema into a library of C++ class definitions. Requirements derived from the STEP specifications and the needs of the VTS are also represented in component libraries; however, these libraries are not automatically generated.

Much of the general functionality needed to support the application model is implemented separately from the application model. This software, represented in the figure as *Core Access Operations*, implements the STEP specifications for EXPRESS and the STEP exchange file format. The other software needs of the VTS, also supported as separate software libraries, are based on analysis of the validation process and users' needs (Morris, 1991a).

[6]C++ has been chosen as the programming language for implementation because it provides the following features:

- good performance in interactive situations,
- programming language constructs which support those found in EXPRESS (i.e. hierarchies and networks of data structures),
- interfaces to externally developed software (specifically, a user interface toolkit *InterViews* (Linton, 1991) and several object-oriented database systems), and
- object-oriented features which enable code reuse.

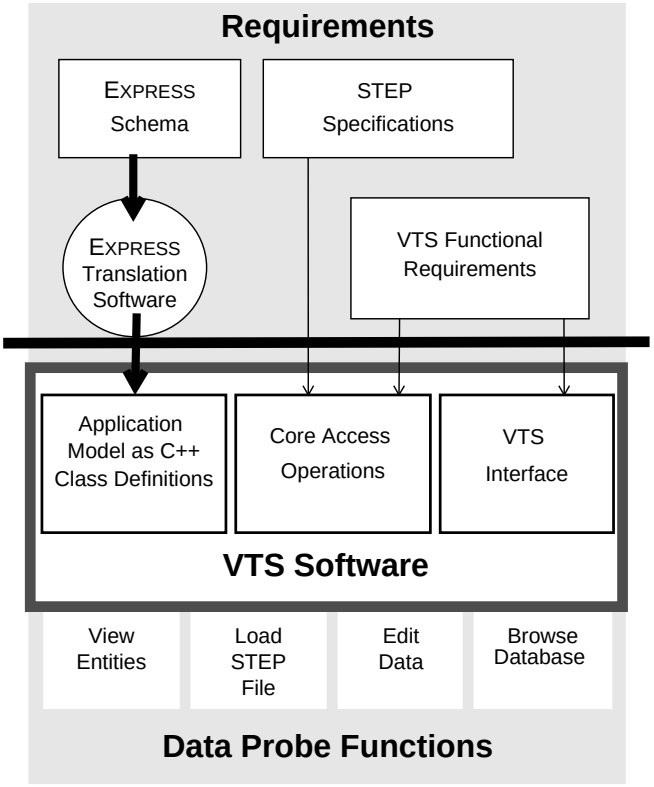

FIGURE 3 : VTS SOFTWARE GENERATION

The component libraries of the VTS software are integrated to support the functional requirements for the validation process and are accessible through a single user interface. The interface provides mechanisms for initiating the functions. A representative sample of these functions is shown in the bottom row of **FIGURE 3**. Together the libraries and the functional interface make up the VTS software. This figure only shows a select subset of the components of the VTS software. The composition of the software is explained in more detail in the remainder of this section and the functions supported by the user interface for the Data Probe tool are described in the following section.

VTS Software Layering

The VTS software libraries can be decomposed into four layers based on specialization with respect to the needs of the VTS system and a specific application model. This design is intended to foster the reuse of code and the integration of external software into the system. **Table 1** shows the software components of each layer. The four layers focus on different functional areas and are named accordingly.

The first three layers represent functionality that is tailored to the VTS needs. Most of these software components have been (or are being) developed for the VTS and are based on externally developed software. The components of the last layer, *generic*

TABLE 1 VTS SOFTWARE LAYERING

VTS Layer	Software Components
Multiple application model	EXPRESS to C++ translator
	Data Converter
Application model specific	Application Model: Schema Class Library
	Application Database
	Data Probe tool
	Translators from CAx, IGES, etc.
VTS specific	STEP Core library
	VTS Interface library
	Data Editor library
Generic systems	Generic User Interface libraries: X Windows, InterViews
	Database management system
	Abstract Data Type libraries
	C++ compiler and standard libraries
	Operating system: POSIX (FIPS151, 1988)

systems, are available from external sources and do not need to be developed specifically for the VTS. The software components are briefly discussed here; the VTS design document (Morris, 1992) provides a more extensive description.

Multiple Application Model Libraries.

The *multiple application model* layer provides the software needed for handling more than one application model. This layer supports the configuration of tools, application models, and test data to support the validation of different application models. In the current system many of these functions are manually controlled. The software components in this layer are external to the VTS run-time environment.[7]

This layer includes the *EXPRESS to C++ translator* (McLay,1990), part of the NIST PDES Toolkit (Clark 1990). The tool automatically translates an application model into the C++ class definitions used to represent it in the VTS software as illustrated in **FIGURE 3**. These class definitions are referred to as the *Schema Class Library.*

[7]The *Data Converter* should ultimately be integrated into the VTS run-time environment; however, this integration is beyond the current scope of the project.

The *Data Converter* converts data to a new version of an application model. It takes as input the data corresponding to an application model and a listing of changes to the application model and outputs data corresponding to the new version of the application model. Initial work has been done in the design of the system and the language for specifying some of the changes to a model (Kohout, 1992) (Clark, 1992).

Application Model Specific Libraries.

The *application model specific* and the *VTS specific* layers contain the data structures needed to support the computerized manipulation of data based on a common schema or application model. The *application model specific* layer represents the components that are tailored to the application model undergoing validation. These components are updated each time the application model changes.

The *Schema Class Library* is the representation of an application model in C++ and provides the data structures, functions, and dictionary specific to that model. Most of the changes to support the validation of a new application model are limited to the *Schema Class Library.* Other libraries in this layer — the *Application Database* and the *Data Probe tool* — only need to be re-installed to reflect their interaction with the new *Schema Class Library.*[8] This design enables the creation of tools which can be tailored to a

particular application model by linking with the library for that model.

The *Application Database* is the database system that has been installed with a particular application model.[9] The schema definition for the *Application Database* is provided by the class definitions in the *Schema Class Library.* The installation process involves preprocessing the class definitions from the *Schema Class Library* to generate the database. The preprocessor is provided by the database supplier. In this way the database can use the same representation form for the data that is used by the other components of the VTS.

Each *Translator* from a specific CAx[10] system is a separate software component. A translator functions by accepting data from the system being translated and creating new instances of the classes in the *Schema Class Library.*

VTS Specific Libraries.

The *VTS specific* layer of the software architecture includes libraries to support the general functional requirements of the VTS software and is independent of the application model being tested. This layer contains software to support the following functional areas:

- data storage,
- user interface,
- data editing and data and schema browsing.

This software is divided into libraries based on its dependencies on STEP or external systems. The *STEP Core* library supports requirements specific to STEP, while the *VTS Interface* and the *Data Editor* libraries support other requirements specific to the VTS tools. This division is illustrated in **FIGURE 3** as separate requirements. Dependencies between the libraries in this layer are minimized to enable their reuse.

The *STEP Core* library provides functionality for supporting the *Schema Class Library* and for accessing a dictionary of information about the application model at run-time. The dictionary contains the names of entities and other descriptive information. Support for the *Schema Class Library* is provided through a set of abstract classes which support the basic constructs found in EXPRESS. These classes form the basis for the *Schema Class Library* and allow the other libraries in this layer to manipulate the contents of the *Schema Class Library* in a general way.

The *Data Editor* library extends the *STEP Core* library to support data editing. This library includes functions for editing instances of product data and functions for manipulating groups of instances. The latter functionality includes merging STEP exchange files, searching sets of instances, and checking sets of instances for completeness with respect to the application model. Some of this functionality may be migrated to the *STEP Core* library should it prove to be more generally useful.

The *VTS Interface* library is designed with the potential for multiple types of display[11]. The strategy for this is to encapsulate the functional content of the interface so that it is not tightly coupled with the display functionality. The initial version of the interface uses the InterViews (Linton, 1991) toolkit. The *VTS Interface* is built on the InterViews library by extending it to support the specific interface of the Testbed tools. The extensions include classes for displaying data instances, groups of data instances, and an application model.

Generic Systems.

The generic systems are completely independent of the specific needs of the VTS and are available as self-contained packages either in the public domain or as commercial systems. The software in this layer isolates the rest of the VTS software from dependencies on a particular hardware platform. The generic systems refer to toolkits or other external software which is directly integrated into the VTS architecture and include the operating system and compiler used in developing the software. Other generic systems used in the VTS software include:

- InterViews user interface toolkit for the X11 (Scheifler, 1989) environment;
- Open Object-Oriented Database Toolkit from Texas Instruments[12] (Wells, 1992); and
- C++ streams library for system i/o.

Summary

The VTS architecture is based on a decomposition of the functional needs for validation of STEP application protocols. The object-oriented techniques of encapsulation and abstraction are

[8]The impact of a change to the *Schema Class Library* on the translators is dependent on the particular changes in the application model. Resolving such changes will probably be more difficult than re-installing the *Schema Class Library*.

[9]This term is used to distinguish it from the generic database system software which is independent of an application model.

[10]CAx is any Computer-Aided operations/processes, including: MCAD (Mechanical Computer-Aided Design), e.g. drawing/drafting; ECAD (Electrical Computer-Aided Design), e.g. PCB layout; MCAE (Mechanical Computer-Aided Engineering), e.g. solids modeling; ECAE (Electrical Computer-Aided Engineering), e.g. logic design; CAM and CIM (Computer-Aided Manufacturing and Computer-Integrated Manufacturing), e.g. NC processing.

[11]The initial version uses the InterViews toolkit. Another potential interface is to an ASCII terminal.

[12]The Open Object-Oriented Database Toolkit from Texas Instruments is a research prototype.

used to define reusable software components which support the functions needed for automating the validation process.

The structure of the software enables code reusability and system extensibility. The software developed for the VTS can provide the foundations for STEP-related systems or general purpose editors for structured information. The architecture defines a structure for attaching new software components as the VTS expands to cover broader functional requirements. The new functions can be integrated into the system without disturbing the existing functionality.

In particular, the *Core* and *Schema Class libraries* can be used for other implementations based on the STEP models or any models written in Express. These libraries represent the data structures for a particular application model. For example, a stand-alone translator for extracting data from IGES (IGES5.1, 1991)[13] to a proposed STEP format for geometry was prototyped using an initial version of these libraries (PDES, 1990b). A translator such as this is not dependent on the *VTS Interface* library which may require a sophisticated windowing system. However, the translator shares the functionality for representing the application model with the VTS software. Therefore, it is able to directly access the VTS database using the interface provided in *Schema Class Library.* The interface to the database is transparent to a translator based on this library and should not need to be updated when the *Schema Class Library* is integrated with a database.

Likewise, parts of the *VTS Interface* and *Data Editor* libraries are useful for any general purpose editor of highly structured information.

THE DATA PROBE TOOL

The Data Probe is the major application which uses the various libraries described. It is intended to allow people involved with the creation and testing of STEP Application Protocols to examine and populate data files corresponding to those models. This section of the paper describes the Data Probe tool and illustrates the user interface to the tool. The examples used in this section are based on the STEP models for geometry and topology[14].

Each Data Probe executable is specific to a particular information model, and new versions of the Data Probe are automatically generated from an EXPRESS schema. The creation of a new Data Probe tool is a three step process. First the EXPRESS schema is translated into a *Schema Class Library*, which is then compiled and linked with the other libraries of the VTS. A UNIX shell script called `mkProbe` automates this process. `mkProbe` accepts an EXPRESS schema as input and outputs a new Data Probe specific to that schema.

The Data Probe tool consists of five types of windows:

1 Data Probe management window (**FIGURE 4**),

2 Entity Type List window (large window in **FIGURE 5**),

3 Entity Instance List window (**FIGURE 6**),

4 STEP Entity Editor (SEE) windows (small window in **FIGURE 5**), and

5 STEP Entity Descriptor (SED) window (**FIGURE 7**).

These windows may be manipulated in the typical ways (move, open, close, hide, expose, resize, etc.) supported by the X windowing system. Any number of temporary SEE and SED windows may be created and destroyed as the editor is being used. A single SEE window is shown in the left side of **FIGURE 5**, partly obscuring the Entity Type List window.

The presentation of these windows is implemented in the VTS Interface library; The underlying functionality is implemented in the Data Editor library; and the data is represented and stored using the Schema Class Library.

Data Probe Management Window

The Data Probe management window provides the user with system functions such as saving or appending files, clearing the Entity Instance List, and quitting the editor. There is also a line for brief messages to the user. In **FIGURE 4** the status window provides feedback to the user that there is one incomplete instance in the list of instances. This particular message would appear when a file is saved or when the user asks to verify the instances. Such operations are initiated using the pull down menu under the title *File Management*.

Entity Type List Window

The Entity Type List window contains a scrollable list of all the entities defined in the EXPRESS schema or schemas used to create the editor. Instances of these entities make up a STEP exchange file. In **FIGURE 5**, the first few entities in this window are *angle_measure*, *area_measure* and *axis1_placement*. The *circle* entity is highlighted in this window because the user has selected it.

At the bottom of the window are two command buttons which indicate the primary use of this window. When the user decides to create an instance of an entity, the window is used to select the type of entity to create. The user selects the name of the entity type to be created (by pointing to the name with the mouse cursor and clicking a mouse button) and then selects the *Create* command button in a similar manner. This causes a SEE window to pop up for a new instance of that type of entity. The example illustrates an instance of “Circle” has just been created.

The second function initiated by this window is the display of type information about a particular entity. As with the create func-

[13]IGES is the Initial Graphics Exchange Specification.

[14]This is the EXPRESS schema from a early draft version of ISO 10303 Industrial Automation Systems — Product Data Representation and Exchange — Part 42: Integrated Generic Resources: Geometric & Topological Representation.

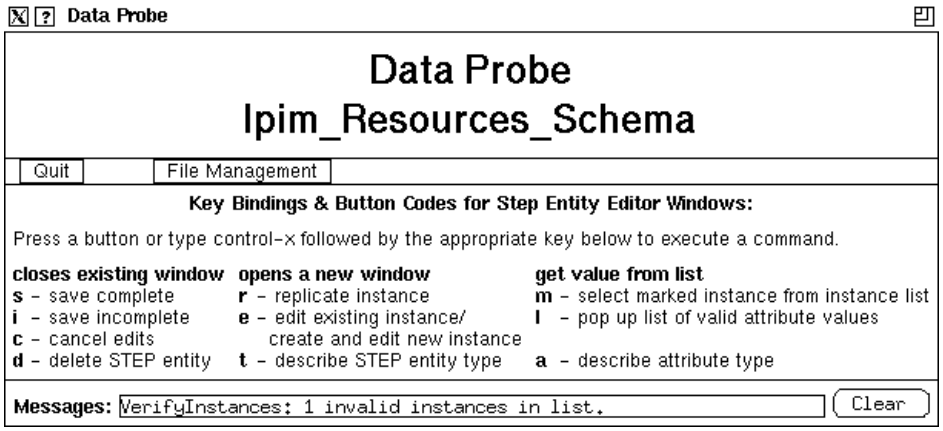

FIGURE 4 : DATA PROBE MANAGEMENT WINDOW

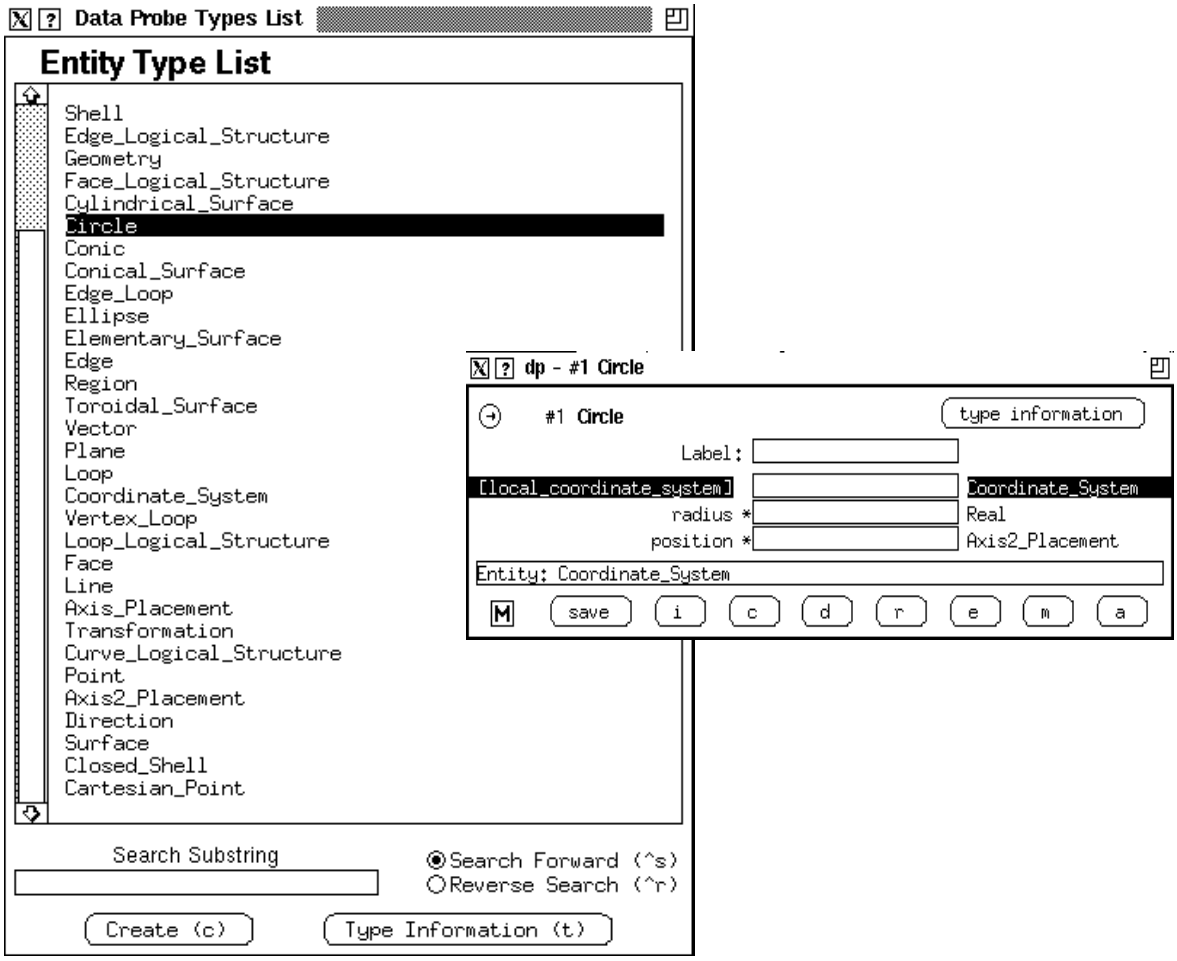

FIGURE 5 : ENTITY TYPE LIST WINDOW

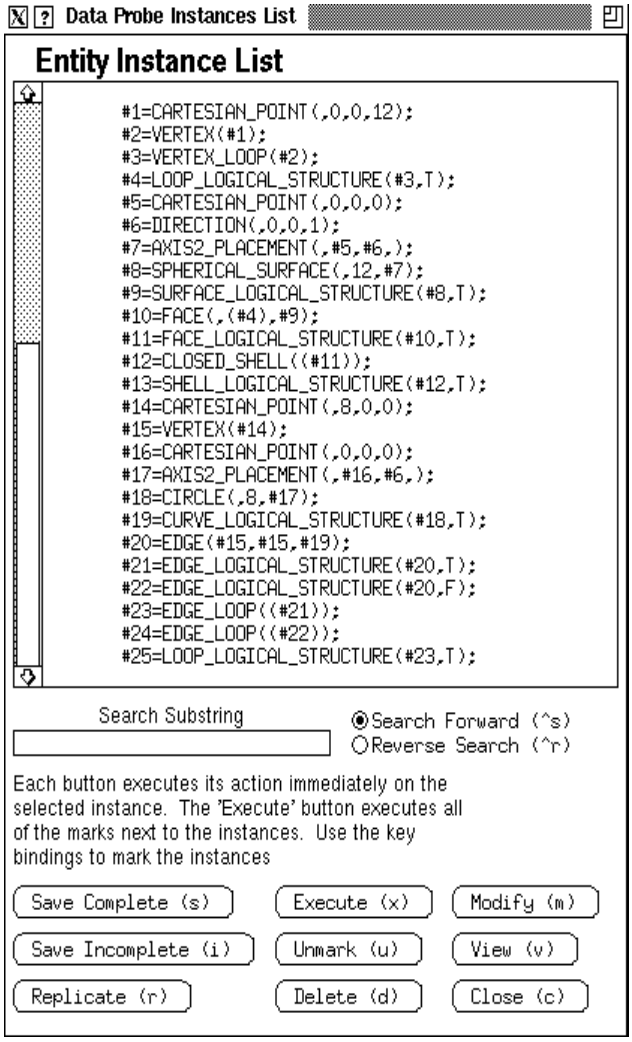

FIGURE 6 : ENTITY INSTANCE LIST WINDOW

tion, the user selects the desired entity type and presses the button labeled *Type Information*. This action brings up a SED for that entity type.

Entity Instance List Window

The Entity Instance List window has two main parts. The top part has the appearance of the "DATA" section of a STEP exchange file. Entity instances are listed, one instance per line. (Lines that do not fit on the screen are truncated.) An indication of the editing status (such as *I* for *Incomplete*) of an instance may be displayed to the left of the instance.

The bottom part of the window contains the controls for a search utility that may be used to find text strings. The bottom part also contains a set of command buttons to perform actions on instances such as: delete, modify, view, save, etc. To use the command buttons, an instance is selected with the mouse, and then the desired command button is selected with the mouse. If either viewing or modifying is selected, a SEE window pops up.

STEP Entity Editor and STEP Entity Descriptor Windows

The SEE window shown in **FIGURE 5** is used to edit the data values for a circle. The circle shown has 3 attributes: *local_coordinate_system*, *radius*, and *position*. The user can enter values for these attributes into the editable fields in the middle of the window.

The STEP Entity Descriptor (SED) window provides more detailed information about an entity type. The SED in **FIGURE 7** give more detailed information about the entity *circle*. This window shows that *circle* is a specialized subtype of *conic*, *curve*, and *geometry*. The attribute *local_coordinate_system* is inherited from the parent supertype *geometry*. [15]

CONCLUSION

The most significant impacts of the VTS software architecture are on the amount of time needed for and on the reliability of the validation of an application protocol. The integration of the software significantly reduces the amount of time needed for validation and improves the reliability of the results. These improvements enable application protocols to be validated more quickly and with less effort.

Several significant features of the VTS architecture support responsiveness to the needs for validation testing. These features include the following:

- a structure for isolating changes to the system due to changing requirements (i.e. the application model, different software or hardware platforms, the evolution of STEP, etc.);
- a single format for data representation, thereby eliminating both the need to translate the data to a variety of formats and the errors associated with such translations;
- a single point of access for the user, thereby reducing the amount of time spent in learning to use the software and minimizing the impact on the user when new functionality is introduced; and
- a collection of reusable source code, which supports the integration of additional functionality into the system and

[15]Multiple inheritance is not yet supported by the VTS software.

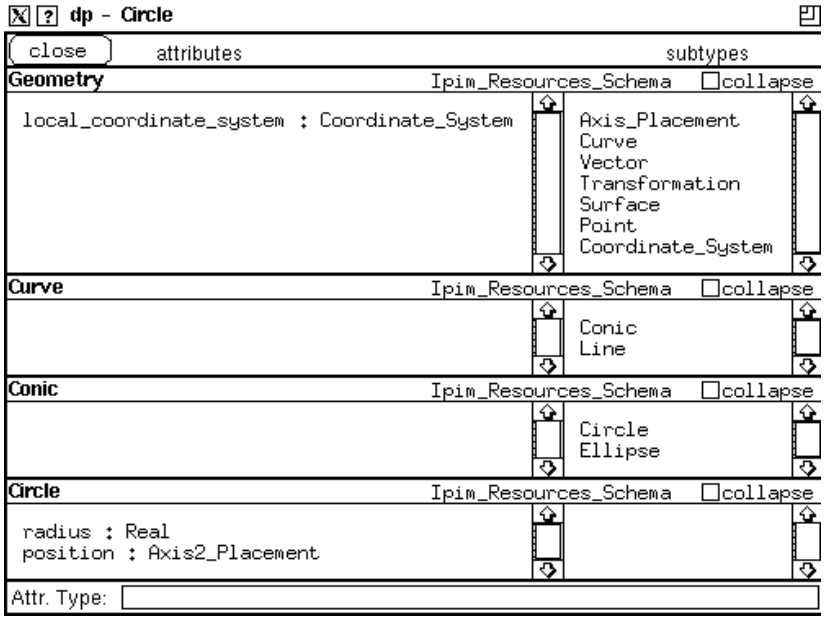

FIGURE 7 : STEP ENTITY DESCRIPTOR WINDOW

also supports the implementation of other STEP-based systems.

Status and Availability of the Software

Several components of this architecture have been implemented to support the needs for STEP application protocol validation within the National PDES Testbed (Mitch, 1990). The *Data Editor*, *STEP Core*, and *VTS Interface* libraries, the EXPRESS to C++ translator for creating a *Schema Class Library*, and the Data Probe tool are in the public domain and are available from NIST. They can be obtained from the ftp site at

ftp.cme.nist.gov

or via modem or mail server (Katz, 1991(Ressler, 1991). Contact the Factory Automation Systems Division (301 975-3508 or npt-info@cme.nist.gov) more information on how to obtain the software, or for other information related to the work at the National PDES Testbed.

Initial implementation of portions of the architecture have already been used by other projects (PDES, 1990b)(McLay, 1990) and future collaborations are planned with the NIST Process Planning Testbed.

Current development activities are addressing the use of TI's Open Object Oriented Database in the system and the use of the STEP Standard Data Access Interface (ISO, 1992d) to allow interoperability with multiple database systems.[16] The architecture defined here provides a basis for these expansions to the VTS software.

REFERENCES

Breese, J. N., McLay, M. and Silvernale, G., Validation Testing Laboratory User's Guide, NISTIR4683, National Institute of Standards and Technology, Gaithersburg, MD, October 1991.

Clark, S. N., An Introduction to The NIST PDES Toolkit, NISTIR 4336, National Institute of Standards and Technology, Gaithersburg, MD, May 1990.

Clark, S.N., Transformr: A Prototype STEP Exchange File Migration Tool, NISTIR 4944, National Institute of Standards and Technology, Gaithersburg, Maryland, October 1992.

Federal Information Processing Standard 151, POSIX: Portable Operating System Interface for Computer Environments, IEEE 1003.1/Draft 12, September 1988.

The Initial Graphics Exchange Specification (IGES), Version 5.1, IGES/PDES Organization, NCGA, Fairfax, VA, September 1991.

International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration— Product Data Representation and Exchange — Overview and Fundamental Principles*, Draft International Standard, ISO TC184/SC4, 1992a.

International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Description Methods: The EX-*

[16]Funding for this work is provided by the Department of Defense's Defense Advanced Research Projects Agency (DARPA).

PRESS Language Reference Manual, Draft International Standard, ISO TC184/SC4, 1992b.

International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Clear Text Encoding of the Exchange Structure*, Draft International Standard, ISO TC184/SC4, 1992c.

International Organization for Standardization, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Standard Data Access Interface Specification*, Working Draft, ISO TC184/SC4, 1992d.

Kohout, R.C., Clark, S.N., Considerations for the Transformation of STEP Physical Files, NISTIR 4793, National Institute of Standards and Technology, Gaithersburg, Maryland, March 1992.

Katz, S., STEP On-Line Information Service User's Guide, NISTIR4491, National Institute of Standards and Technology, Gaithersburg, MD, January 1991.

Linton, M,. InterViews Reference Manual Version 3.0-alpha, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Silicon Graphics, January 1991.

McLay, M.J. and Morris, K.C., The NIST STEP Class Library, C++ at Work-'90 Conference Proceedings, September 1990. (Reprinted as NISTIR 4411.)

Mitchell, Mary, Development Plan: Validation Testing System, NISTIR 4417, National Institute of Standards and Technology, Gaithersburg, MD, October 1990.

Mitchell, M., A Proposed Testing Methodology for STEP Application Protocol Validation, NISTIR 4684, National Institute of Standards and Technology, Gaithersburg, MD, September 1991.

Mitchell, M. J., Morris, K. C. The Use of Application Model Validation in Testing a Proposed Standard, *Proceedings of the Sixth Annual ASME Database Symposium - Engineering Data Management: Key to Integrated Product Development*, American Society of Mechanical Engineers, New York, August 1992.

Morris, K.C., McLay, M. and Carr, P. J., Validation Testing System Requirements, NISTIR 4676, National Institute of Standards and Technology, Gaithersburg, MD, September 1991a.

Morris, K.C., Mitchell, M.J. and Sauder, D. Validating STEP Application Protocols at the National PDES Testbed, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, November 1991b.

Morris, K. C., Sauder, D., Ressler, S., Validation Testing System: Reusable Software Component Design, NISTIR 4937, National Institute of Standards and Technology, Gaithersburg, MD, October 1992.

PDES, Inc. Test Report for Context-Driven Integrated Model (CDIM) Application A1, Skeels, J., ed., PDES, Inc. internal report PMG012.01.00, SCRA, Charleston, SC, April 1990a.

PDES, Inc., Block Point Release 2.1 Systems Manual PTI018.01.00, Silvernale, Gerard, ed., SCRA, Charleston, SC, February 1, 1990b.

Ressler, Sandy, "The National PDES Testbed Mail Server User's Guide", NISTIR 4508, National Institute of Standards and Technology, Gaithersburg, MD, Jan., 1991.

Scheifler, R.W., X Protocol Reference Manual for Version 11, O'Reilly and Associates, Inc., Sebastopol, CA, 1989.

Stroustrup, B., ANSI X3J16/90-0020, C++ Language System Reference Manual.

David Wells, José Blakeley, Craig Thompson, "Architecture of an Open Object-Oriented Database Management System," *IEEE Computer*, October 1992.