
National PDES Testbed



Architecture for the Validation Testing System Software

Katherine C. Morris

U.S. DEPARTMENT OF
COMMERCE
Technology Administration

National Institute of
Standards and Technology
John W. Lyons, Director

December 1991



Architecture for the Validation Testing System Software

Katherine C. Morris

Abstract

This document is part of the National PDES Testbed Report Series and is intended to complement the other reports of the Validation Testing System (VTS) project. Other documents describe the methodology for model validation and software requirements for automating that methodology.

The problem of sharing data has many facets. The need for the capability 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 business environment.

This document describes an architecture for an integrated software environment to support the validation of STEP application protocols. The architecture provides a basis for software development to support the Validation Testing System within the National PDES Testbed. (PDES, Product Data Exchange using STEP, is the U.S. effort in support of the international standard.) The software architecture and the use of object-oriented techniques enables code reusability and system extensibility. The software developed for the VTS can provide the foundations for STEP related systems or software projects requiring general purpose editing tools for structured information.

Table of Contents

	Abstract	iii
	Table of Contents	v
	List of Figures	v
	List of Tables	v
1	Introduction	1
2	Background	2
	2.1 The Need for Validation Testing Software	2
	2.2 Validation Testing Software at the National PDES Testbed	3
3	The VTS Software Design Strategy	7
	3.1 Establishing the Testing Environment	8
	3.2 VTS Software Layering	10
	3.3 Relationships between the VTS Software Components	15
	3.4 Summary	17
4	Conclusion	17
5	References	19
APPENDIX A	VTS Document Series	21

List of Figures

FIGURE 1	Interim Software at the National PDES Testbed	5
FIGURE 2	The Validation Testing System Software Environment	6
FIGURE 3	VTS Software Generation	9
FIGURE 4	Dependencies between VTS Software Components	15

List of Tables

TABLE 1	VTS Software Layering	11
---------	-----------------------------	----

1 Introduction

The emerging Standard for the Exchange of Product Model Data, a project of the International Organization for Standardization and commonly referred to as STEP¹, addresses the need to share data in a complex computer environment. STEP will provide a basis for a common understanding of and communication of data, thus allowing it to be shared. STEP will include 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.

The information models within STEP are integrated and organized into *application protocols*. An application protocol contains the information model, in a computer readable format, for a specific application area. This document describes an architecture for software to support validation of information models such as those contained in application protocols [Palmer91] or other application models², which may be proposed standards within STEP.

The architecture will be used as the basis for software implementation for the Validation Testing System (VTS) [Morris91b] at the National PDES Testbed³. Section 2 of the document contains a background discussion of validation software, both present and future, at the National PDES Testbed. Subsequent sections describe and discuss the design of the VTS software architecture. Further details of the methodology used for validation [Mitch91] and requirements for automating that process [Morris91a] are described in other documents in the VTS documentation series (Appendix A).

The audience for this document includes software designers, developers, and project managers. The document provides an overview of the software architecture for the VTS. 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 VTS software.

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* [ISO1].

2. The term *application model* will be used throughout this paper to refer to the domain specific information model which is being evaluated -- whether that be an application protocol, an application resource model, a context driven integrated model (CDIM), or some other form of an application schema.

3. The National PDES Testbed is located at the National Institute of Standards and Technology. Funding for the project 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.

2 Background

This section gives 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 other documents in the VTS series (see Appendix A.) An overview of the need for validation software is presented here. A discussion follows of the software, which is currently being used at the National PDES Testbed for validation of application protocols, and of directions for future software implementation.

2.1 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 which supports that 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 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 [Mitch91] 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 increases the productivity of the people validating the models by an order of magnitude, if not more, depending on the quality of the tools. The project schedule for the validation of an application model was severely impacted by the lack of reliable and efficient tools [PDES90].

Furthermore, 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 model did not require objects to be founded in geometric space, which caused 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 -- which had gone undetected, in the review process which relied on visual inspection, for over two years.

2.2 Validation Testing Software at the National PDES Testbed

The software to support validation simulates the data access requirements of an application area⁴. 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 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 complete if the tests are not computer processable and repeatable.

4. Other software requirements for validation [Morris91a] 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.

In addition, the software will assist in managing the complexity of the information involved in the testing process. The information involved includes

- product data associated with an application model,
- relationships within the data,
- the application model as defined in Express [ISO11], and
- the English description of the application model.

The VTS software will provide functions to assist in preparation of product data to be used in testing an application model. The software will support the editing, browsing, and formatting of product data, and the browsing of the application model, in addition to support for the actual tests.

The National PDES Testbed has been used for STEP validation testing since 1989. The current software to support the testing process, the *interim system*, consists of a set of tools which are not integrated. The interim system, as illustrated in **Figure 1**, is described in the *Validation Testing Laboratory User's Guide* [Breese91]. The tools in this system were collected from a variety of sources, and, as a result, they operate on a variety of hardware platforms and in a variety of software environments. The current method of sharing data throughout the testing process is by exchanging data files between the different tools.

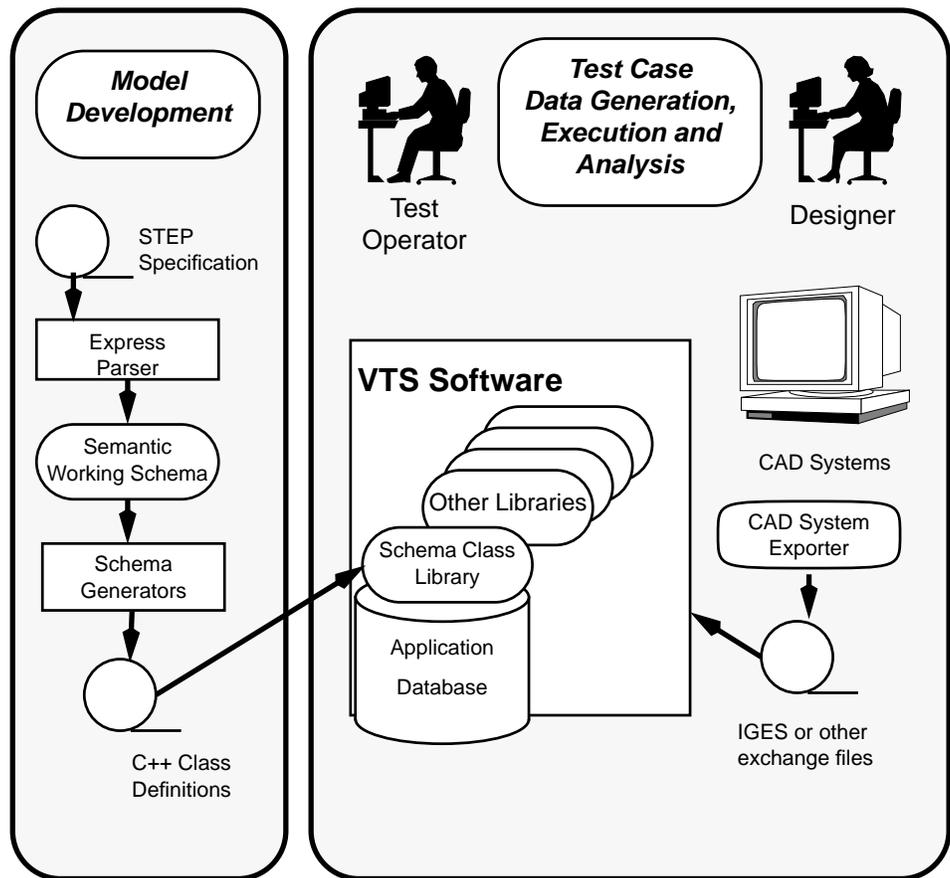
The future VTS software will provide an integrated set of tools for automating the validation testing process. The integration of the tools will provide a more efficient environment. The VTS software will improve on the existing system in the following areas:

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

The interim system requires data translation, which can introduce errors or inconsistencies, every time data is moved between activities in the testing process. Moreover, the translation process and the associated manual steps, such as importing and exporting the data, are time consuming. With respect to data editing, the interim system does not track which portions of the data are complete and which need further development. Manual support for this function is extremely difficult, given the amount of data, and leads to inconsistencies in the data. In addition, the manual configuration of the different versions of all the intermediate data files is error prone. The future environment, illustrated in **Figure 2**, will allow the users to operate in the same environment and through a common interface, rather than with each tool separately, throughout most of the testing process.

The future environment will use a database system, rather than exchange files, as the primary means of data sharing⁵. Currently data is assembled from the various tools and is manually integrated using exchange files to create a single data set. This manual process forces 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.

FIGURE 2 The Validation Testing System Software Environment



The remainder of this document describes the VTS software architecture. The VTS software is composed of reusable software libraries which support the different functional areas for the validation process. These libraries are combined to provide an end-user tool which supports the needs for the validation process. The architecture is designed to

5. Note that this environment will not preclude the use of exchange files as a means of importing and exporting data into and out of the system. Exchange files will also be used as the primary means of sharing data until a database has been integrated into the system

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 focus for implementation will be to replace the existing data editor, which is the weakest tool in the existing system. Once the software needed to support this set of functions is complete, the development process will focus on integrating a database system into the software. In the meantime, users will have access to the new editor. The integration of the database system will be transparent to the users; however, the integration will enable the expansion of the functions supported by the end-user tool.

3 The VTS Software Design Strategy

The VTS software architecture integrates modular software libraries. These libraries will be incorporated into a single system which supports functions needed for the validation process. The use of object-oriented techniques and standard interfaces will enable software reusability. The system will use as much software as is available from external sources. When such software is unavailable, the necessary software will be developed. Specifically, support for the implementation methods specific to STEP will need to be developed.

Since STEP is a developing standard, the mechanisms for its implementation are not stabilized. Furthermore, since these mechanisms are developing concurrently with the application models which the VTS software is used to validate, it is unlikely that externally developed software will be available in the time frame needed. These mechanisms include the specification language for the application models -- Express [ISO11] -- and the data interface formats, such as the exchange file format [ISO21]. The VTS must be responsive to changes in these implementation mechanisms by providing software which is quickly and easily adaptable to new versions of the mechanisms.

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

- 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 deal with the structure of the VTS software and are discussed throughout this section.

The validation process involves the computerized manipulation of data based on the application model being evaluated. Therefore, the architecture for the VTS software includes a library of data structures and access functions for representing that model. These data structures must be rich enough to store the data and to provide the semantic

information from the application models at runtime. Semantic information needed at runtime includes the names of the structures, their fields, and, when possible, acceptable values for the fields and rules governing the relationships between instances.

3.1 Establishing the Testing Environment

An application model is represented in many formats throughout the validation process: English, Express and Express-G [ISO11], 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 [Clark90]. In the interim system each tool uses a different computer format; however, in the VTS software a single format will be used: C++⁶ [Stroustrup90]. The C++ representation of the application model can be automatically generated from an Express description.

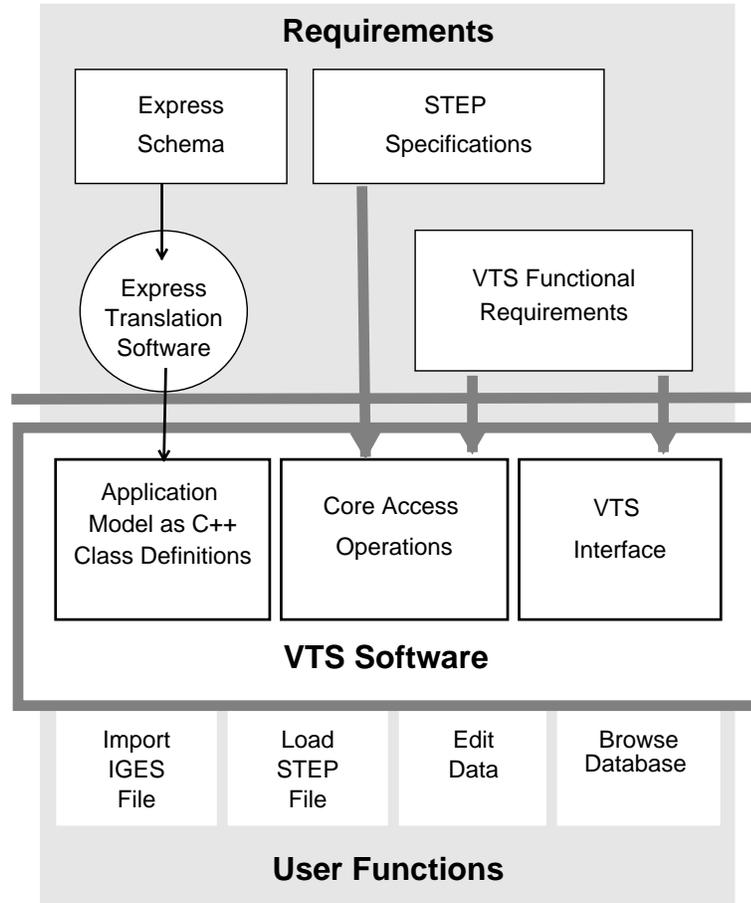
Figure 3 illustrates the VTS software development methodology for producing a testing environment for a specific application model. An application model, as represented in a library of data structures and access functions, is integrated into the VTS software. The application model, described in Express, is translated into a software library as shown in the upper left hand corner of **Figure 3**. This library is then installed in the VTS software to generate a new testing environment for that model. 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.

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 mirror those found in Express (i.e. hierarchies and networks of data structures),
- interfaces to externally developed software (specifically, a user interface toolkit *InterViews* [Linton91] and several object-oriented database systems),
- in the process of becoming standardized, and
- object-oriented features which enable code reuse.

FIGURE 3

VTS Software Generation



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 direct translation of an Express schema into a library of C++ class definitions is the representation of the application model used by the software. Requirements derived from the STEP specifications and the needs of the VTS are also represented in component libraries; however, these libraries can not be 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 [Morris91b] and users' needs [Morris91a].

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 sharable

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 following sections.

3.2 VTS Software Layering

The VTS software libraries can be decomposed into five 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 five layers focus on different functional areas and are named accordingly:

- multiple application model libraries,
- application model specific libraries,
- VTS specific libraries,
- generic systems, and
- machine dependent systems.

The first three layers represent functionality which is tailored to the VTS needs. Most of these software components need to be developed for the VTS, but they can make use of externally developed software. The components of the last two layers, *generic* and *machine dependent systems*, are mostly available from external sources and do not need to be developed specifically for the VTS. For the most part the external systems are included into the VTS software through libraries. But some of the components, such as the operating system or utilities provided with a database system, are separate systems.

This section defines the VTS software components. The components in the three middle layers are especially significant to the software design. The relationships between the components in these three layers is discussed in the following sub-section. All of the software components will be described in further detail in the VTS design document (Appendix A).

TABLE 1

VTS Software Layering	
VTS Layer	Software Components
Multiple application model libraries	Configuration System Data Converter Express Parser: Fed-X
Application model specific libraries	Application Model: Schema Class Library Application Database Data Probe library Translators from CAx ⁷ , IGES, etc.
VTS specific libraries	STEP Core library VTS Interface library Data Editor library Express Browser library
Generic systems	Generic User Interface libraries: X Windows, InterViews Database management system Abstract Data Type libraries Generic Graphic Interfaces: Hoops ⁸ , PEX
Machine dependent systems	C++ compiler and standard libraries Operating system: POSIX [FIPS151]

3.2.1 Multiple Application Model Libraries

The *multiple application model* layer provides the software needed for handling more than one application model. This layer supports the transition and 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.⁹

7. 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 and photoplotting.

8. No approval or endorsement of any product by the National Institute of Standards and Technology is intended or implied.

The *Configuration System* governs which executable programs can be used with which data files. This involves tracking which version of an application model was used to generate the C++ representation used in the software and which data files correspond to these application models. The *Configuration System* should be accessible by an executing program. Until this software is in place, its function is performed manually.

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 [Kohout90].

Also included in this layer is the NIST PDES Toolkit [Clark90]. The program *fedex_plus* [McLay90], which is part of the toolkit, automatically translates an application model into the C++ class definitions used to represent it in the VTS software as illustrated in **Figure 3**. The output of the program is the *Schema Class Library* in the application model specific layer described below.

3.2.2 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 which 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 and functions 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 Library* -- only need to be re-installed to reflect their interaction with the new *Schema Class Library*.¹⁰ 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 term *Application Database* refers to a database system which has been installed with a particular application model. This term is used to distinguish it from the generic database system software which is independent of an application model. The schema definition for the *Application Database* is provided by the class definitions in the *Schema Class Library*. The application database is dependent on the commercial database management system chosen for use in the Testbed. 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.

The *Data Probe* library composes the functions encapsulated in the other libraries to assemble an end-user tool. The tool provides data editing and browsing functions with the appropriate user interface and application model. The tool also supports browsing of

9. The *Configuration System* and *Data Converter* should ultimately be integrated into the VTS run-time environment; however, this integration is beyond the current scope of the project.

10. The impact of the change in the *Schema Class Library* on the translators is dependent on the particular changes in the application model.

the application model. The *Data Probe* library is initialized to use the *Schema Class Library* for a particular application model.

Each *Translator* is a separate software component. A translator functions by accepting data from its input system (the system being translated) and creating new instances of the classes from the *Schema Class Library*.

3.2.3 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:

- user interfaces,
- data editing and browsing functions, and
- browsing the application models written in Express.

The software in this layer is divided into libraries based on its dependencies on STEP or external systems. The *STEP Core* library and the *Express Browser* library support 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 as separate requirements in **Figure 3**.

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 display¹¹. The strategy for this is to encapsulate the functional content of the interface so that it will not be tightly coupled with the display functionality. The initial version of the interface uses the InterViews [Linton91] toolkit. The *VTS Interface* is built on this 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. This library is also responsible for interfacing to the operating system such as when locating, opening, and closing files.

11. The initial version uses the InterViews toolkit. Another potential interface is to an ASCII terminal and would be based on a curses library.

The *Express Browser* library contains the functions needed for browsing Express models. The library would support both a hypertext style interface for browsing entity definitions in the Express source text and a graphical display of the application model based on Express-G.

3.2.4 Generic and Machine Dependent Systems

The generic and machine dependent 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 these two layers isolates the rest of the VTS software from dependencies on a particular platform. The generic systems refer to toolkits or other external software which is directly integrated into the VTS architecture. The lowest layer, machine dependent systems, refer to the operating system and compiler used in developing the software.

Generic User Interface Toolkits, such as libraries provided with the InterViews toolkit, will be used as a basis for the *VTS Interface* library in the X11 [Scheifler89] environment.

Generic Graphic Interface Toolkit systems, such as Hoops [Wiegand89] or PEX [PEX] will be used to support the three dimensional graphical display of products, which is planned as future work.

The *Database Management System* to be used in the Testbed has not yet been chosen. A set of requirements for this system in the Testbed is available [Morris90a]. The primary requirement is that the system has a C++ interface and is capable of using class definitions written in C++ as its data definition language.

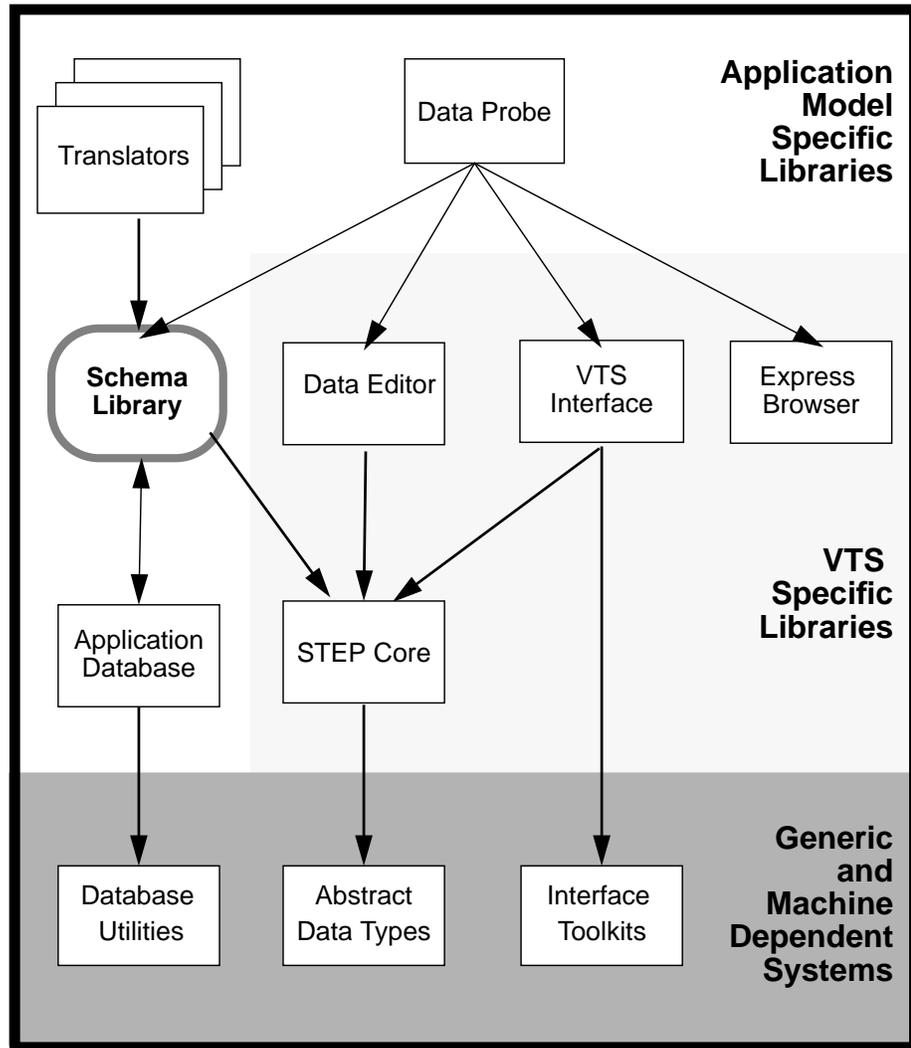
Abstract Data Type libraries contain C++ classes for representing primitive data types such as strings, integers, and real numbers, complex data types such as lists and arrays, common system functions such as file management, and other general purpose utilities such as for handling binary trees or regular expressions. It is anticipated that some of these libraries will come from the provider of the database management system and/or C++ compiler; others may need to be developed.

3.3 Relationships between the VTS Software Components

This section discusses the relationships between the VTS software components from the three middle categories of the software layering shown in **Table 1**: the application model specific libraries, the VTS specific libraries, and the generic systems. The VTS software design is confined initially to the components in these layers. Dependencies between the software in these layers are shown in **Figure 4**.

FIGURE 4

Dependencies between VTS Software Components



→ Indicates Dependency
 ○ Generated Library

The division of the VTS software into component libraries enables code reuse by encapsulating the functionality needed for different aspects of the validation process. Several considerations are influential in defining the components of the VTS software, but two functional requirements are of particular importance:

- the need to easily transition the software to a new application model, and
- the desire to enable different user interfaces to be developed.

The VTS software is conceptually divided based on these requirements. The information from a particular application model is encapsulated in the *Schema Class Library*. The *Data Editor* library encapsulates the information and operations needed for manipulating and editing that information. The *VTS Interface* library contains the operations needed to present that information in a general way to the user. The presentation format in the user interface is not influenced by the specific content of the information but uses generalizations, or *abstractions*, about the structure of the information. These abstractions are based on Express and are supported by the *STEP Core* library. (This particular division of libraries is represented in the illustration of the VTS software in **Figure 3**.)

The VTS architecture layering supports modularization of the software. Component libraries in each layer are dependent on some of the libraries within the same layer or in the next, more general layer, as shown in **Figure 4**; however, the more general layers are not dependent on the less general layers. For example, the *VTS specific* layer depends on the *generic systems* layer, but not vice versa. In general, dependencies are limited to the interface between layers. The exception is in the area of the database management system. The database system will come from external sources and will have its own particular structure. Illustrated here is an external view of that system.

The VTS architecture is structured to enable reuse of the software. A different style of user interface could be developed for the VTS software by replacing the *VTS Interface* library without affecting the data editing and representation capabilities of the system. For example, two different systems could be developed to support window-based and ASCII-based interfaces by replacing this library. These systems would provide the same functions in terms of editing and representational capabilities, since those components of the software would not change.

Parts of the VTS software will be useful to other programs. In particular, the *Core* and *Schema Class libraries* represent the data structures for the application model and can be used by other implementations based on STEP models. For example, a stand-alone translator for extracting data from IGES [IGES5.1]¹² to a proposed STEP format was developed using an initial version of these libraries [PTI2.1]. 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 will be 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. Therefore, the translator will 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* will be useful for any general purpose editor of highly structured information.

12. IGES is the Initial Graphics Exchanges Specification.

3.4 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 used to define reusable software components which support the functions needed for automating the validation process. The software is decomposed in two ways:

- common functionality is abstracted into reusable software libraries creating a functional layering of the software; and
- software components are specialized with respect to specific functional requirements of STEP and of the VTS.

The structuring 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.

4 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 will significantly reduce the amount of time needed for validation and improve the reliability of the results. These improvements will enable application protocols to be validated more quickly and with less effort.

Several significant features of the VTS architecture allow the software to be responsive 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, STEP, ...);
- a single format for data representation, thereby eliminating 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 end user, thereby reducing the amount of time spent in learning to use the software; and
- a collection of reusable source code, which will support the integration of additional functionality into the system and also support the implementation of other STEP-based systems.

The structure provided by the architecture allows the software to be easily transitioned to validate a new application model. The structure also minimizes the impact of other changes in the environment on the system. The integration of the software into a single system minimizes the errors introduced into the process by frequent translation of the data. The impact on the users when the system is expanded to support new functions is also minimized. Furthermore, the architecture provides a structure for developing reusable source code which will allow additional related functions to be supported more quickly and provide a natural mechanism for integrating these into an existing system.

In addition, the software developed may benefit other projects as well. Initial implementation of portions of the architecture have already been used by other projects [PTI2.1][McLay90].

The VTS project will implement components of this architecture to support the needs for STEP application protocol validation within the National PDES Testbed [Mitch90]. Implementation of the software will include the identification of existing software to address the functional areas needed and the development of software to support functions not available externally. The architecture defined here provides a basis for developing and integrating this software.

5 References

- [Breese91] 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.
- [Clark90] Clark, S. N., An Introduction to The NIST PDES Toolkit, NISTIR 4336, National Institute of Standards and Technology, Gaithersburg, MD, May 1990.
- [FIPS151] Federal Information Processing Standard 151, POSIX: Portable Operating System Interface for Computer Environments, IEEE 1003.1/Draft 12, September 1988.
- [IGES5.1] The Initial Graphics Exchange Specification (IGES), Version 5.1, IGES/PDES Organization, NCGA, Fairfax, VA, September 1991.
- [ISO1] ISO 10303 Industrial Automation Systems -- Product Data Representation and Exchange -- Part 1: Overview and Fundamental Principles, Working Draft N43 ISO TC184/SC4/WGPMAG, Mason, H., ed., October 7, 1991.
- [ISO11] ISO 10303 Industrial Automation Systems -- Product Data Representation and Exchange -- Part 11: Description Methods: The EXPRESS Language Reference Manual, Committee Draft N14 ISO TC184/SC4, Spiby, P., ed., April 29, 1991.
- [ISO21] ISO 10303 Industrial Automation Systems -- Product Data Representation and Exchange -- Part 21: Clear Text Encoding of the Exchange Structure, Committee Draft ISO TC184/SC4, Van Maanen, J., ed., March 12, 1991.
- [Kohout90] Kohout, Robert, STEP Data Translations, internal documentation, National Institute of Standards and Technology, Gaithersburg, MD, September 1990.
- [Linton91] 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.
- [McLay90] McLay, M.J. and Morris, K.C., The NIST STEP Class Library, C++ at Work-'90 Conference Proceedings, September 1990. (Reprinted as NISTIR 4411.)
- [Mitch90] Mitchell, Mary, Development Plan: Validation Testing System, NISTIR 4417, National Institute of Standards and Technology, Gaithersburg, MD, October 1990.

- [Mitch91] Mitchell, M., A Proposed Testing Methodology for STEP Application Protocol Validation, NISTIR 4684, National Institute of Standards and Technology, Gaithersburg, MD, September 1991.
- [Morris91a] Morris, K.C., McLay, M. and Carr, P. J., Validation Testing System Requirements, NISTIR 4676, National Institute of Standards and Technology, Gaithersburg, MD, September 1991.
- [Morris91b] 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 1991.
- [Palmer91] Palmer, M. and Gilbert, M., Guidelines for the Development and Approval of STEP Application Protocols, ISO TC184/SC4 N1 Version 0.7 working draft, January 25, 1991.
- [PDES90] Test Report for Context-Driven Integrated Model (CDIM) Application A1, Skeels, J., ed., PDES, Inc. internal report PMG012.01.00, SCRA, Charleston, SC, April 1990.
- [PEX] PEX Protocol Encoding, Version 5.0P, X Public Review Draft, Barry, S.C., ed., September 1990.
- [PTI2.1] Silvernale, Gerard, Block Point Release 2.1 Systems Manual PTI018.01.00, PDES, Inc., SCRA, Charleston, SC, February 1, 1990.
- [Scheifler89] Scheifler, R.W., X Protocol Reference Manual for Version 11, O'Reilly and Associates, Inc., Sebastopol, CA, 1989.
- [Stroustrup90] Stroustrup, B., ANSI X3J16/90-0020, C++ Language System Reference Manual.
- [Wiegand89] Wiegand, Gary, HOOPS Reference Manual, 2nd ed., Ithaca Software, Ithaca, NY, 1989.

No approval or endorsement of any product by the National Institute of Standards and Technology is intended or implied.

APPENDIX A VTS Document Series

This document complements others in the National PDES Testbed Report Series which provide detailed technical information relating to the Testbed software. Those documents which specifically address aspects of the Validation Testing System are described below.

Validation Testing Systems Plan lays out the tasks and the overall approach for the initial implementation of the Validation Testing System. (NISTIR 4417)

Proposed Testing Methodology for STEP Application Protocol Validation describes the complete process used to develop and validate application protocols. This methodology document focuses on the analysis of application models and planning for validation testing. (NISTIR 4684)

Validating STEP Application Models at the National PDES Testbed describes a strategy for automation based on an analysis of the information flow in the application protocol development and testing process, and on initial experiences with automation for validation testing at the National PDES Testbed. (NISTIR 4735)

Validation Testing System Requirements describes functional requirements for automation of the VTS. This document also provides an overview of the VTS software environment. Requirements for the VTS system are driven by the STEP development effort and reflect the needs of the National PDES Testbed users. (NISTIR 4676)

Architecture for the Validation Testing System Software describes an architecture for software which supports the testing of information models for validity and correctness. The architecture provides a basis for software development within the National PDES Testbed. (NISTIR 4742)

Validation Testing System Software Design provides guidelines for the implementation of the VTS software. The document describes an architecture for creating and integrating software libraries to support the tools for the VTS. Designs for the components of the software are also provided. (forthcoming)