DiagML - AN INTEROPERABILITY PLATFORM FOR TEST AND DIAGNOSTICS SOFTWARE

Eric Gould, DSI International, (714) 637-9325, egould@dsiintl.com
Danver Hartop, DSI International, (714) 637-9325, dhartop@dsiintl.com
Erick Lee, XML Designs, (714) 429-0037, erick.lee@xmldesigns.com
Ion A. Neag, TYX Corporation, (703) 264-1080, ion@tyx.com
Mark Wilson, XML Designs, (714) 429-0037, mwilson@xmldesigns.com

ABSTRACT

The integration of diagnostic software and test execution environments developed by different vendors requires a common data exchange format. The paper describes a new format based on XML, called the Diagnostic Modeling Language (DiagML). This format is currently used for integrating software applications developed by DSI International and TYX Corporation. The design-to-test process supported by DiagML is illustrated through a diagnostic application for a simple analog circuit.

Keywords: diagnostics, test, data exchange, language, XML

1 INTRODUCTION

The implementation of the design-to-test paradigm requires the exchange of information between diagnostics applications and test execution environments. Because these classes of applications are frequently developed by different software vendors, their interoperability requires a common data exchange format.

Although existing diagnostic data exchange formats, such as the AI-ESTATE standard [1] developed under the aegis of the IEEE, may be well suited for data interchange between diagnostic reasoners, their lack of explicit accommodations for test parameters make them less than ideal for the exchange of data between diagnostic environments and test executives.

Moreover, recent advances in software technology (componentization and distribution technologies, the Extensible Markup Language (XML) and related technologies [2] [3], etc.) support a dramatic increase in software development productivity, provide excellent support for software interoperability and make possible new classes of distributed applications. Unlike the sometimes drawn-out process involved in the production of official standards, the design and standardization process associated with these technologies is extremely fast, being driven by the immediate needs of the software industry. The resulting formats often become de facto standards due to widespread usage and support.

In 2001 a consortium of companies decided to build on these technological advances, using the lessons learned from their involvement in other standardization processes, for developing a new

XML-based data exchange format named DiagML (acronym for "Diagnostic Modeling Language) [4]. This paper presents the design of DiagML, its benefits and its integration in software products developed by DSI International and TYX Corporation.

2 XML BENEFITS

The XML language and the related family of technologies provide significant benefits for data exchange, compared to traditional modeling languages:

1. XML is a platform-independent text format, supporting internationalization and localization.
2. XML is modular and extensible, allowing the definition of new formats by combining and reusing other formats.
3. XML is license-free and well supported by high-quality development environments and by free software components for parsing and generation.
4. XML has gained wide industry acceptance and has been incorporated into a wide variety of services encompassing multi-platform security, distribution and communication.
5. XML has been extended to support databases, data embedding, automatic translation between formats and is at the core of the new web services technology for interoperability.

3 DiagML EVOLUTION PROCESS

The evolution of DiagML is controlled by a working group [5], functioning as a centralized body through which problems are identified and new features are introduced. The working group maintains versioning information and supports the creation of standard tools. Membership to the working group can be requested at the DiagML web site [5]. The DiagML Advisory Committee (which currently consists of representatives from the three charter companies) mediates the efforts of working group members to ensure that the representation of data is consistent with the needs of all users. Currently, the most recent version of the DiagML schema is only available to members of the working group. DiagML will soon become a published (open) format, at which time the advisory committee, as well as the working group, will open itself to new members.

4 DiagML AND AI-ESTATE

The AI-ESTATE standard [1] developed by the IEEE defines a data exchange format based on the EXPRESS language. The industry has so far been reluctant in adopting AI-ESTATE due to its high complexity, the slow pace of the standardization process and the relative difficulty of processing the EXPRESS format.

Furthermore, AI-ESTATE's standardized approach to storing attribute data (in which each data field not explicitly defined within the standard must be handled through a custom extension) is less than ideal in situations where a large number of test parameters (the precise details of which may not be able to be defined in advance) must be passed between diagnostic applications and test execution environments. The only alternative would be a proliferation of non-standard uses of the standard in which individual parametric values can only be exchanged by tools that are "privy" to the customized extensions of the AI-ESTATE format. In short, although it may be well suited for the interchange of data between different diagnostic applications, AI-ESTATE's tight semantics are a hindrance to its use as an interchange format between diagnostic generation and test execution environments. Proponents of AI-ESTATE, of course, might well object that this is a "straw man" argument, since data standardization implies tight semantics and, because AI-ESTATE was not conceived as a method of passing test parametric data between applications, the standard would clearly not include data definitions for the parameters required by test executives. This objection, of course, only underscores the need for a common format for the representation and interchange of both diagnostic and test data between diagnostic and test environments.

The inherent extensibility of the XML language has resulted in the development of many simple, yet sophisticated mechanisms for the storage of generic data. Rather than legislate the specific attributes that can be represented within a particular "flavor" of XML, these techniques allow large numbers of attributes (using a wide variety of data types) to be represented within an XML file. This is precisely the approach that is needed to represent the design attributes and test parameters that must often be passed between diagnostic development and test execution environments, since the specific parameters that will be required for each type of test procedure may not always be easily enumerated in advance. XML schemas can be used to standardize data representation, yet allow the necessary flexibility in certain areas (such as parameter definition) where the precise data requirements may not be known ahead of time.

5 DiagML DESIGN

5.1 DiagML File Structure

A DiagML file has the structure shown in Figure 1, containing four main sections:

1. **DesignData**: information related to the Unit Under Test (UUT) model, including UUT components and pins, functions and failure modes
2. **MaintenanceData**: definition of adjust/replace actions to be performed according to the result of the diagnosis
3. **TestData**: definition of tests, including test location and test parameters
4. **DiagnosticData**: test strategy information, including sequence information and parametric data

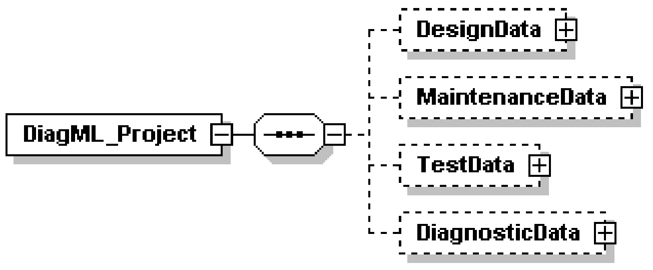


Figure 1. Structure of a DiagML File, High-Level View

Several areas of the DiagML design, considered relevant for the context of the present paper, are described in the following sections.

5.2 Test Data

Figure 2 shows the DiagML model for test data. A **TestProcedure** element represents an executable function or procedure implemented in a software component (such as: a function in a DLL, a method in a COM component, an ATLAS PROCEDURE, etc.). The **ExecutionEnvironment** parameter identifies the software environment that is required for executing the test procedure (for example, LabWindows/CVI, TYX PAWS, etc.), while the **Locator** identifies the software module that stores the executable test procedure (for example, the file name for DLL; the ProgID for a COM component, etc.). The definition of a test procedure also contains the possible values of the **Outcome** it returns, as well as the input and output **Parameters** supported by the test procedure.

A **Test** element represents the application of a test procedure at a specified **Location** in the model (e.g. a UUT pin). The definition of a test also contains a set of **Attributes** and information about the **Coverage** of the test in terms of functions and failure modes.

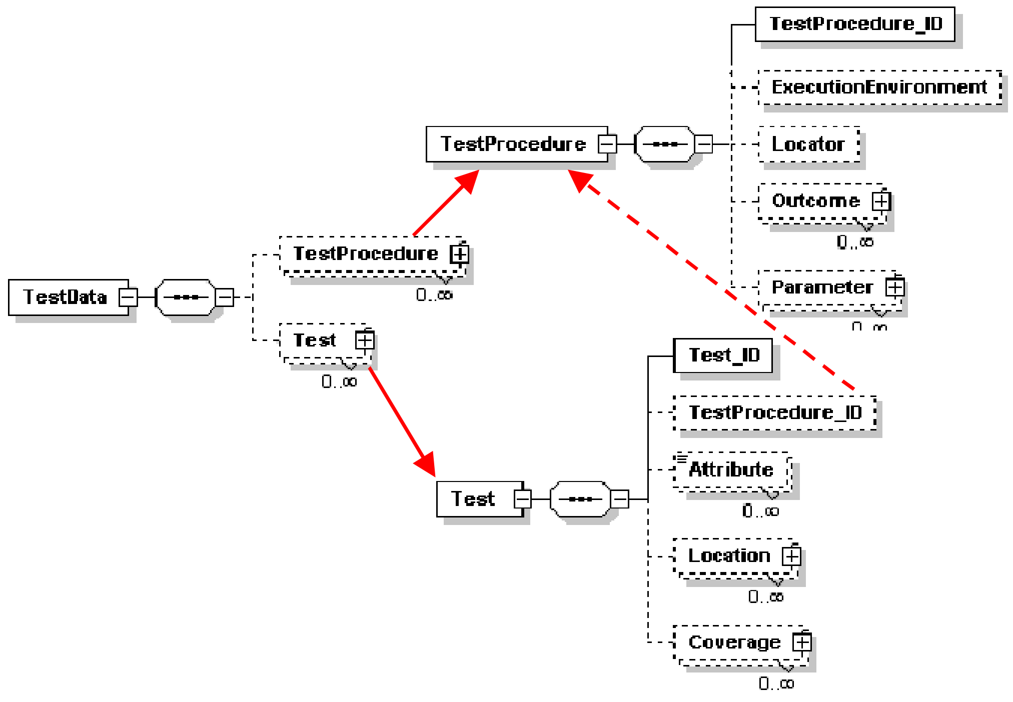


Figure 2. Test Data in DiagML (partial representation)

5.3 Diagnostic Data

The **DiagnosticData** section of a DiagML file may contain one or more **DiagnosticStrategy** elements. A diagnostic strategy is an executable entity that may be used to diagnose a specific UUT. Each diagnostic strategy includes one or more **DiagnosticProcedure** elements. Diagnostic procedures may invoke other diagnostic procedures (similar to the functions of a C program). A diagnostic procedure contains a number of **DiagnosticStep** elements (shown in Figure 3), each of them representing one of the following operations:

- **TestExecution**: execution of a test, defined in the **TestData** section
- **DiagnosticProcedureExecution**: call to another diagnostic procedure, defined within the same diagnostic strategy
- **MaintenanceProcedureExecution**: execution of a maintenance procedure, defined in the **MaintenanceData** section
- **End**: termination of the diagnostic procedure

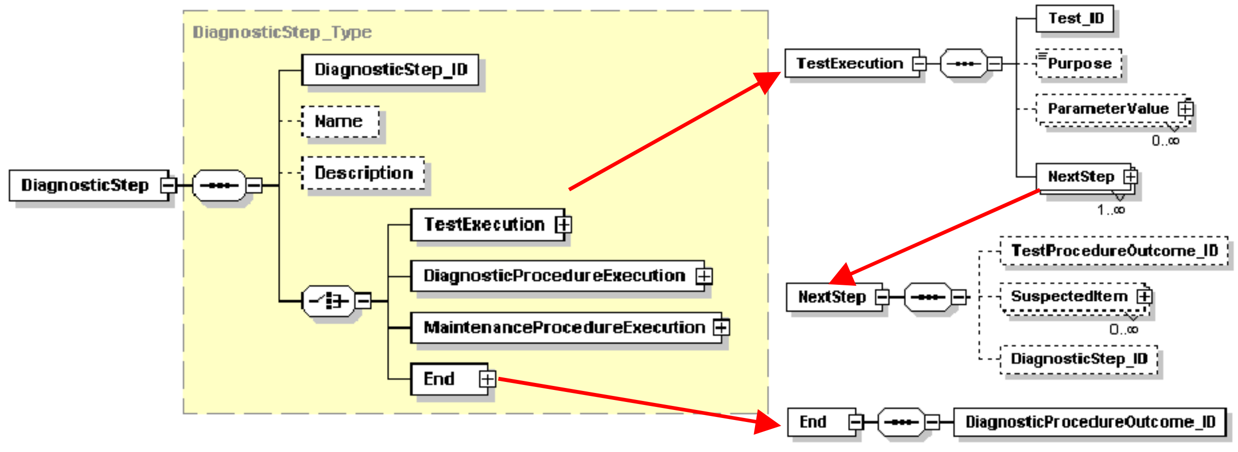


Figure 3. Diagnostic Data in DiagML (partial representation)

The DiagML representation of a test execution step (element **TestExecution** in Figure 3) specifies the test to be executed and the values of input parameters to be passed to the test procedure assigned to that test.

The DiagML model for diagnostic steps contains sequencing information, indicating the next step(s) to be executed. In the case of a test execution step (see Figure 3), the DiagML file specifies, for each possible outcome returned by the test procedure, the next diagnostic step to be executed. This information is contained in **NextStep** elements, along with data indicating the suspected item(s) identified for the particular outcome.

When finished, diagnostic procedures may return an outcome, indicating the diagnostic conclusion. This information is contained in the **End** elements.

5.4 Parametric Data

The DiagML representation for the value of a parameter is shown in Figure 4. A **PrimitiveValue** type may be one of the primitive data types supported by the XML Schema specification, a group (the equivalent of the C data type `struct`), or an array. A **CustomValue** element contains the hex-encoded serialized value, along with the Programmatic Identifier (ProgID) of a

COM component that may be used to de-serialize and edit the custom data type. This design enables the exchange of data between diagnostic software and test execution environments that support the "plug-in" integration of specialized data editors.

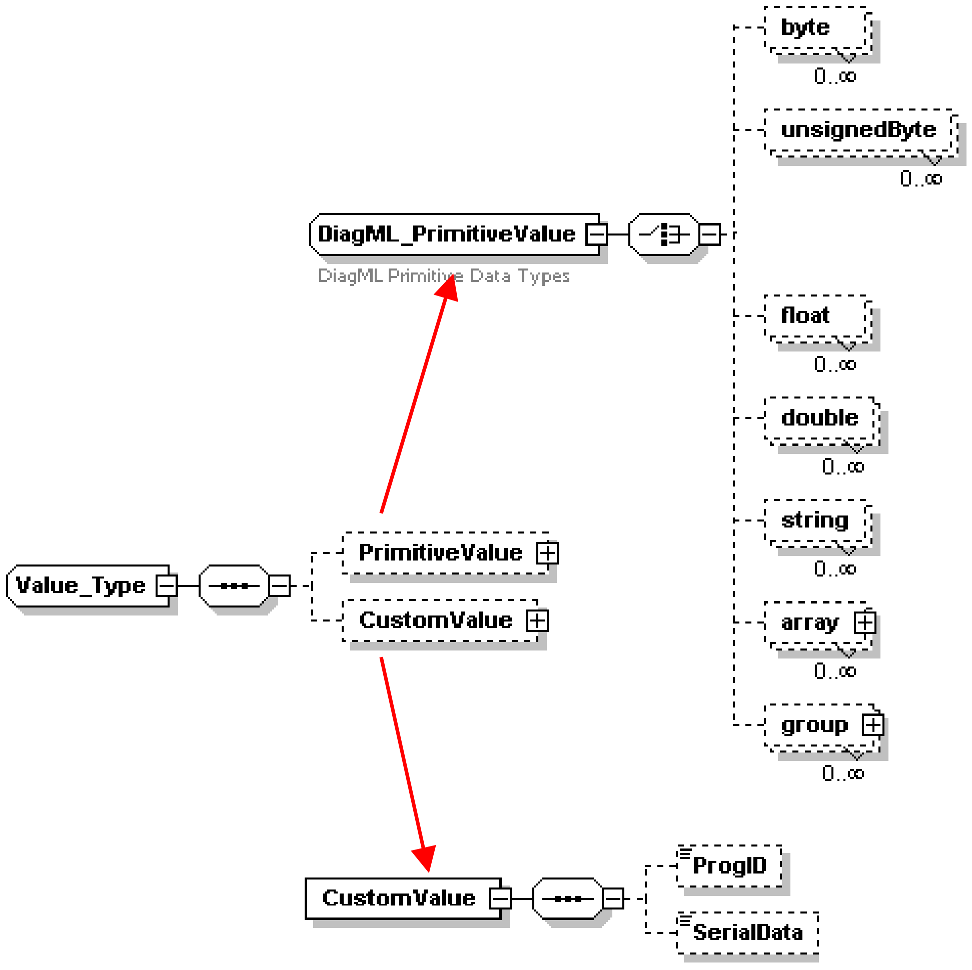


Figure 4. Parametric Data in DiagML (partial representation)

6 DiagML APPLICATION

6.1 Integration Architecture

DiagML is currently used for exchanging test and diagnostic data between software applications developed by DSI International and TYX Corporation. The integrated architecture, shown in Figure 5, supports a comprehensive design-to-test development process.

DSI eXpress is a model-based Diagnostics Engineering and System Governing tool. Its object-oriented approach to full-system design capture enables it to analyze the interrelationships between all components of a system and its environment (including the testing environment).

This unified approach provides the platform upon which analysis and optimization can occur throughout all phases of development.

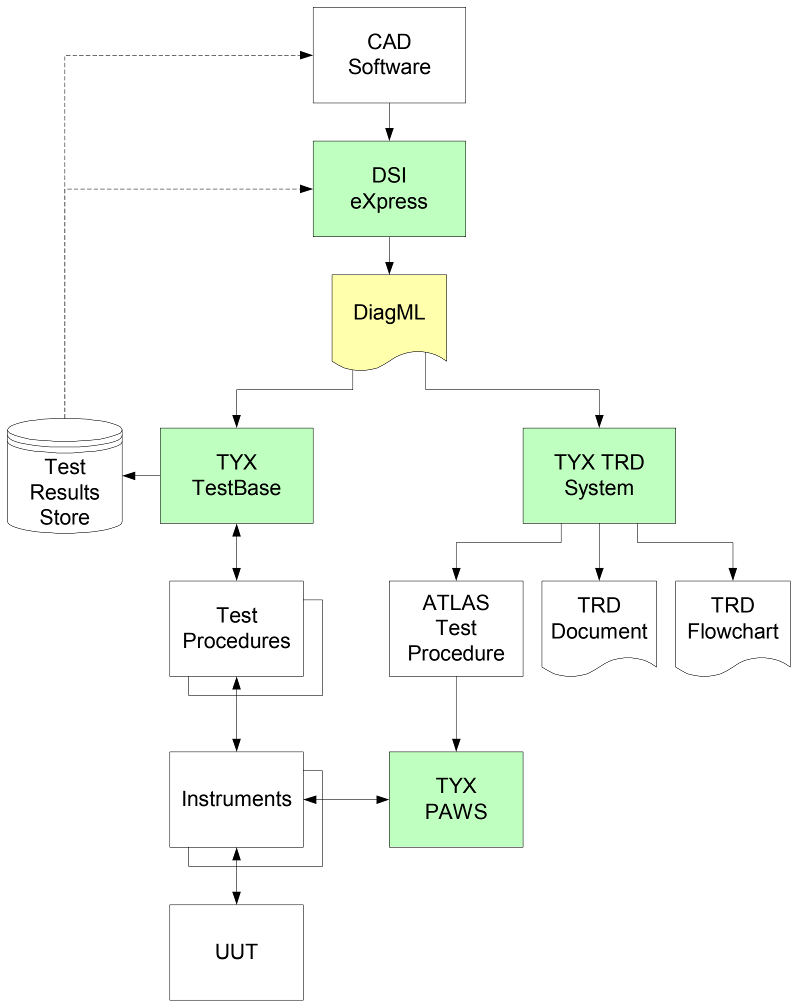


Figure 5. Integration of eXpress, TestBase and the TYX TRD System using DiagML

The eXpress development environment enables the user to build a model of the UUT, possibly importing design data from third-party **CAD Software,** and to define a suite of tests from which a comprehensive diagnostic strategy can be built. eXpress's strategy generation supports both multiple fault and common cause assumptions, different scenarios, operating modes and mission profiles. The strategies produced are directly exportable to DiagML, which supports their transfer to a test executive without loss of diagnostic data.

TYX TestBase [6] is a test executive that supports the visual design and run-time execution of test strategies[1]. TestBase is able to control the execution of external **Test Procedures** developed

[1] TestBase uses the term "test strategy" for the entity called "diagnostic strategy" in eXpress. These two terms are used interchangeably in the present paper.

with multiple programming languages and test environments, including C++, Visual Basic, LabWindows/CVI and ATLAS.

TestBase imports DiagML files, converting them into test definition data and "fault tree" test strategies, stored in its internal databases. The development environment of TestBase allows the user to manually edit the imported test strategies, or to combine them with other manually defined diagnostic procedures. TestBase supports the run-time execution of imported test strategies, by invoking external test procedures (which correspond to the tests defined in eXpress). A test procedure typically generates stimulus signals, measures the UUT response, compares measurement results with predefined limits and returns PASS/FAIL outcome. This outcome is used by TestBase to determine the next test to be executed, to further isolate the fault. This decision is based on the sequence information imported from DiagML. At the end of a test strategy execution TestBase identifies a fault or a group of faults.

TestBase is capable of storing test and diagnostic results in databases or XML files, represented in Figure 5 by the generic block **Test Results Store**. Historical diagnostic information can be used to analyze the frequency and distribution of UUT faults, providing feedback for design improvements. Furthermore, this information may be used for assessing the effectiveness of diagnosis, supporting the improvement of the eXpress model.

The **TYX TRD System** is a documentation system for signal-based Test Procedure Sets (TPSs). The TRD System imports DiagML files, generating Test Requirements Documentation (TRD) files stored in an internal format. If necessary, the user may edit the imported TRD information manually, via a specialized TRD Editor. This information is used by the TRD System to generate TRD documents, test strategy flowcharts and ATLAS test procedures. The ATLAS test procedures may be executed using the **TYX PAWS** software.

6.2 Experimental Results

To demonstrate the use of DiagML in the design-to-test process, TYX and DSI have jointly developed a simple diagnostic application, described in the following. The demonstration UUT, shown in Figure 6, is a simple analog circuit that models the operation of an amplifier, including the decrease of AC gain with frequency and the limitation of the output voltage excursion. Several switches simulate component faults.

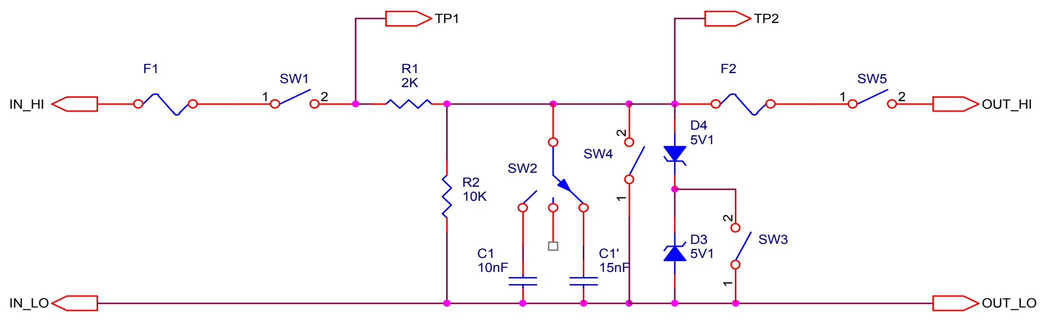


Figure 6. Schematic of the demonstration UUT

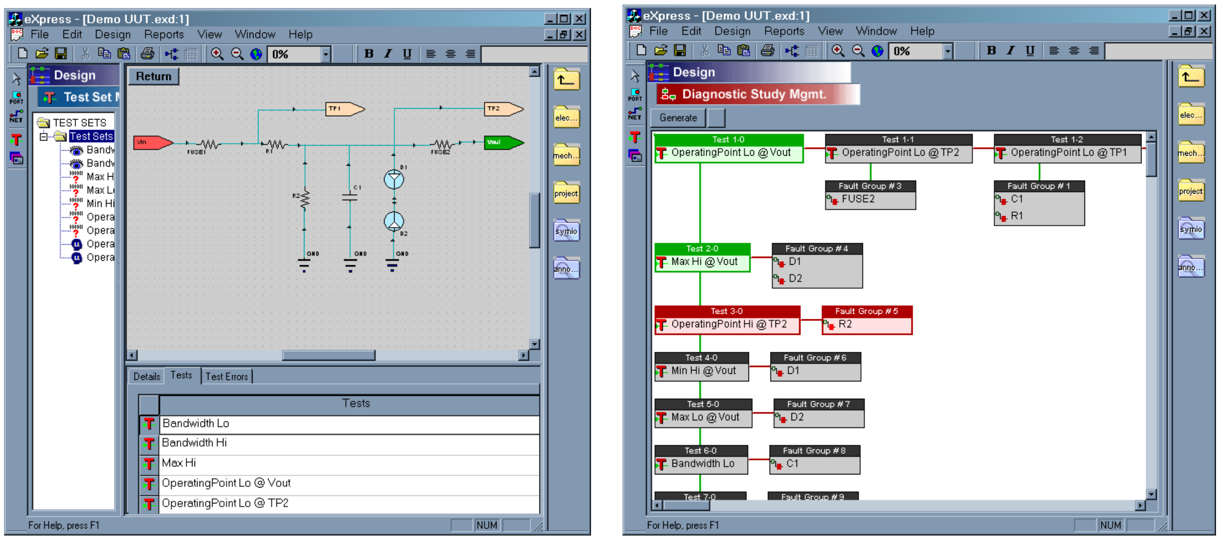
The steps of the design-to-test process used for demonstration are described below.

1. Build an eXpress model for the UUT, defining the ports of the UUT, its components, their interconnection, the failure modes of components and the functions that model the propagation of faults through interconnections.

2. Define a test set in eXpress. The following categories of tests were defined for the demonstration UUT: operating point measurement at three test points; measurement of Max and Min voltage excursion limits at the output; bandwidth measurement through AC sweep.

3. Specify attributes for each test. For example, two attributes of each test represent the High and Low limits for the measurement results. In the case of the demonstration UUT, these limits were calculated through Monte-Carlo analysis in a SPICE simulator. The attributes defined in eXpress are transferred via DiagML to TestBase.

4. Generate a diagnostic strategy in eXpress and evaluate the quality of fault isolation by checking the size of ambiguity groups. If the ambiguity groups are too large, the diagnosis may be improved by adding new test points or new types of tests. Figure 7 shows the eXpress model of the demonstration UUT (along with the test set) and a partial view of the diagnostic strategy generated by eXpress.



	Tests
T	Bandwidth Lo
T	Bandwidth Hi
T	Max Hi
T	OperatingPoint Lo @ Vout
T	OperatingPoint Lo @ TP2

Test 1-0 OperatingPoint Lo @ Vout
Test 1-1 OperatingPoint Lo @ TP2
Test 1-2 OperatingPoint Lo @ TP1
Fault Group # 3 FUSE2
Fault Group # 1 C1 R1
Test 2-0 Max Hi @ Vout
Fault Group # 4 D1 D2
Test 3-0 OperatingPoint Hi @ TP2
Fault Group # 5 R2
Test 4-0 Min Hi @ Vout
Fault Group # 6 D1
Test 5-0 Max Lo @ Vout
Fault Group # 7 D2
Test 6-0 Bandwidth Lo
Fault Group # 8 C1
Test 7-0
Fault Group # 9

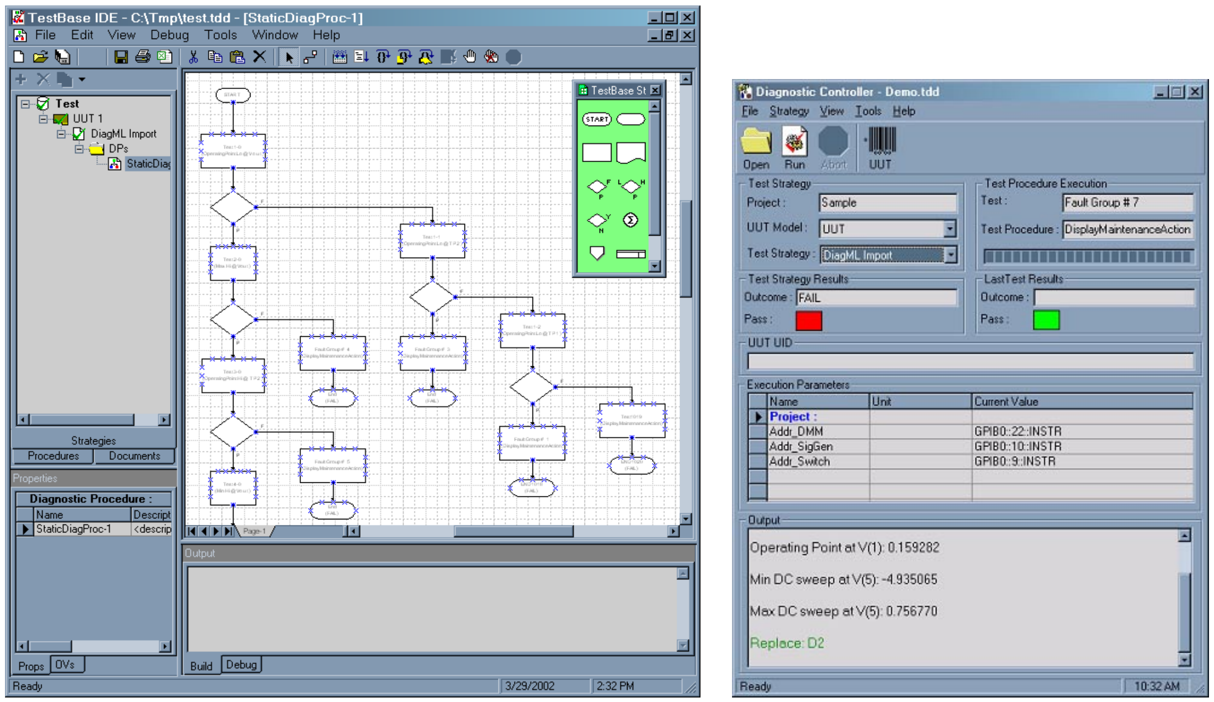
Figure 7. eXpress Model of the Demonstration UUT (left) and Diagnostic Strategy (right)

5. Export the eXpress diagnostic strategy in a DiagML file.

6. Develop TestBase test procedures for the test set defined in eXpress. The following test procedures were developed for the demonstration UUT: Operating Point; Max; Min; Bandwidth. The test equipment used by these test procedures includes a function generator, a digital-analog converter (DAC), a digital multi-meter (DMM) and a switch.

7. Import the DiagML file in TestBase as a test strategy. If necessary, modify the imported strategy or edit its parametric data. Compile the test strategy, making it ready for execution by the TestBase Diagnostic Controller.

8. Validate the proper operation of the test strategy, by executing it while different UUT faults are simulated through switches. Figure 8 shows the test strategy imported in the TestBase Integrated Development Environment and the results of an execution of this test strategy by the TestBase Diagnostic Controller.



Test Strategy
Project : Sample
UUT Model : UUT
Test Strategy : DiagML Import

Test Procedure Execution
Test : Fault Group # 7
Test Procedure : DisplayMaintenanceAction

Test Strategy Results
Outcome : FAIL
Pass : 🟥

LastTest Results
Outcome :
Pass : 🟩

UUT UID

Execution Parameters

Name	Unit	Current Value
Project :		
Addr_DMM		GPIB0::22::INSTR
Addr_SigGen		GPIB0::10::INSTR
Addr_Switch		GPIB0::9::INSTR

Output

```
Operating Point at V(1): 0.159282

Min DC sweep at V(5): -4.935065

Max DC sweep at V(5): 0.756770

Replace: D2
```

Figure 8. Test Strategy Imported in TestBase (left) and Execution Results (right)

Because all the necessary attributes were originally defined in eXpress, manual data input in TestBase was not necessary. This capability is very important for the effectiveness of the design-to-test process, where multiple iterations may be needed as the UUT model changes during the development process. The above capability is enabled by the ability of DiagML to store attribute data, as described in Section 4.

7 CONCLUSION

DiagML is an open XML-based file format that supports the exchange of test and diagnostic data between software applications developed by different vendors.

Because DiagML is based on XML, it can be easily processed by applications developed with diverse software technologies. Furthermore, the future evolution of DiagML is supported by a flexible design and an effective maintenance process.

The design of DiagML aims to provide generic support for test, diagnostics and simulation tools, while enabling future extensions of the model. The current modeling capabilities of DiagML are demonstrated by its utilization for supporting the integration of three commercial products developed by DSI International and TYX Corporation.

8 REFERENCES

[1] *Draft Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*, IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems, IEEE, 2002

[2] *** *Extensible Markup Language (XML)*, World Wide Web Consortium (W3C), http://www.w3.org/XML/

[3] *** *XML Schema*, World Wide Web Consortium (W3C), http://www.w3.org/Schema/

[4] *** "What Is DiagML", *DiagML Web Site*,

[5] *** "DiagML, the Diagnostics Markup Language", *DiagML Web Site*,

[6] *** *"TestBase Features"*, TYX Corporation, 2001