Design of a Flexible, Integrated Testing System for STEP and OMG Standards

KC Morris and David Flater
National Institute of Standards and Technology[1]
U.S.A.

Abstract

New software standards supporting integration of manufacturing and engineering systems are emerging at a rapid pace. Two groups, the OMG (the Object Management Group) and the community producing STEP (formally known as ISO 10303: the Standard for the Exchange of Product Model Data), dominate in the production of standards for manufacturing and engineering industries. Their standards are based on common methods, which can be exploited in developing tests for systems supporting the standards. This paper describes the methods employed and a system that builds on those methods to support the automatic and rapid development of conformance tests for the emerging standards.

Keywords: conformance testing, manufacturing, software, standards

1. Introduction

The more flexibility and accessibility that we achieve in interfaces to system components the higher the demand on the components becomes. The openness of today's software environment has raised expectations for the level of integration that can be achieved. In more "traditional" systems, by which we mean tightly integrated systems where all components were supplied by the same vendor, a uniformity of design dictated the structure. A traditional system demanded a common source language, a shared file system, and a common operating system. These were basic immutable requirements for most systems. Components were designed to operate within those constraints.

Today's software environment is much more flexible. A system typically involves multiple programming languages, hardware and operating systems, and a variety of data storage formats. Components are now designed with a new set of requirements. Commercial off-the-shelf components (COTS) are typically available on a number of platforms with interfaces written in a number of programming languages. Shared file systems are used nearly everywhere.

The result is that COTS must be very adaptable to remain viable and systems integrators must create equally adaptable systems. Systems integration today is a difficult task because of the variability of components. One problem arising for systems in the new environment is that not only do interfaces need to be adaptable across the spectrum of traditional factors (programming languages, operating systems, hardware platforms...) but a new factor is emerging which was not as evident in the older integration tasks: the world view. Different systems use different, possibly incompatible abstractions to model or refer to things in the real world, and these differences lead to integration difficulties.

Standards make the operational environment more manageable. Application program interfaces (API) are joining the ranks of traditional computing standards (such as for programming languages and operating systems) to support systems integration. Success of manufacturing systems integration depends on interface specifications[2] that support a cohesive world view. A cohesive world view does not mean that components for which the interfaces are specified share the same world view but rather that the world views need to come together in a meaningful way. The interfaces are the points of coherence. Testing components for compliance to an interface is one means of testing whether the components are capable of supporting the world view of the system. Testing in this environment often can be as difficult as the integration task itself. However, features of the standards can be used in the creation of a flexible test system as will be described here.

---

[1] Contribution of the National Institute of Standards and Technology is not subject to copyright protection.

[2] In this paper we use the term *specification* to refer to an interface specification, unless stated otherwise. An interface specification does not constrain the internal implementation details of a function.

This paper examines standards for supporting systems integration in the manufacturing area (specifically STEP and OMG, see below) and describes a reusable test system based on the standards. Section 1 describes the role of standards; section 2 overviews the methods used for the standards; section 3 reports on a prototype test system which builds on those methods; and section 4 discusses the usefulness of the testing approach.

1.1 The OMG and STEP standards

Standards provide guidelines for systems integrators. Together, the families of standards from OMG (the Object Management Group) [14] and from the STEP committee (ISO Technical Committee 184, Subcommittee 4, abbreviated ISO TC184/SC4) [18] cover most of the existing domain for manufacturing systems integration. Yet these standards are subject to the same forces of entropy and change that challenge all modern manufacturing enterprises.

The OMG and STEP families share some similarities and have some notable differences. Both have chosen a uniform, albeit different, approach to design. Both define infrastructural, as well as domain-specific, standards. Both address a vast scope. Both use a modularized approach to deal with the scope. The modularized approach has left them in the position of having a series of dependent specifications in which many of the standards in each family point off to other family members at the risk of pointing to something beyond their control.

Although OMG standards are more object-oriented than STEP standards, the most important distinction between the two families is in the types of interfaces that they specify. The OMG specifications focus on functionality or behavior; the STEP specifications focus largely on information "content." Stated another way, OMG specifications are primarily in the computational viewpoint, while STEP specifications are primarily in the information viewpoint [9]. The STEP specifications were conceived at a time when data exchange was a primary integration mechanism. The OMG standards are addressing today's more flexible environment. Given the origins in data exchange, the authors of STEP took to heart lessons learned in needing to be unambiguous in the definition of content and in specifying a complete context for exchange. The result is a very thorough specification; however, many now view these attributes of the standard as unnecessary baggage. OMG interfaces tend to be more simplified views of an underlying system omitting much of the detail necessary for data exchange. On the other hand, the OMG process is now struggling with issues of integration and conformance, which have been initially addressed for STEP.

Many standards developers are grappling with how to bring the rich heritage of STEP into the Internet-based world. STEP contains some features to support its use in what has become known as a *data sharing* (vs. an *exchange-based*) environment [11], but those features are limited in scope and focus on low-level access to data. STEP's Standard Data Access Interface (SDAI) [7] is the first attempt at providing a standard, interactive interface for STEP data (by which we mean an interface that can be accessed conveniently by application software). SDAI combined with a domain specific model, many of which are included in STEP, creates an interactive interface to a particular type of data. The Manufacturing Domain Task Force [12] within OMG produces another style of interactive interface that focuses less on data and more on functionality and is characterized as supporting coarser-grained access. The initial interface [15] from this group borrows largely from STEP for much of the semantic context of the interface. These approaches, along with the exchange-based approach, address different classes of integration problems.

1.2 Standards support for portability

Standards come in a wide variety. "Plug and play" capability is often a goal of a standard; however, given the state of the computing infrastructure on which software is built, seamless integration of software components is still a future vision. Instead we can talk about degrees of portability:

- Design portability: the design of the application software is preserved such that the software can be recoded in a different programming language or using a different API to meet the needs of a particular system configuration.
- Compile-time portability: the source code for the application is preserved such that the program only needs to be recompiled to be integrated into a new system.
- Link-time portability: the compiled source code is preserved such that the application will work in a new system simply by linking with external software.
- Run-time portability: the application can be inserted into a new system "as is" with no modifications to the executable code; some set up of the applications environment may be needed.
- Plug-and-play: with plug-and-play an application is automatically configured to the parameters of the system and can be inserted at run-time.

While plug-and-play would be the best choice in an ideal world, often portability of design is an acceptable alternative given the trade-offs. Unfortunately, design portability is in some ways the most difficult to achieve since people attempt to reuse designs in contexts with different application requirements and semantics. Compile-time, link-time, and run-time portability can be checked in mechanical ways; however, design inconsistencies address the semantics of an interface, are subtler, and can be difficult to detect. It is therefore important to have interfaces that are clearly specified in order for systems to be integrated correctly and easily.

2. Overview of methods

The methods used in defining standards are critical to achieving the different types of portability. This section provides an overview of the methods used by both OMG and STEP and how they are used to specify domain-specific interfaces. As summarized in the table below each standard includes a language and a diagramming technique for specifying content, an architecture for describing the relationship of the different specifications, a family of complimentary specifications on which to build, and specifications for realizing an interface in a programming language. The methods used by both standards support varying degrees of portability depending on the demands of the application.

[Table 1]

2.1 OMG

Domain-specific interfaces in OMG use the following OMG-defined methods:

- The Interface Definition Language (IDL) [1]: a language for specifying interface prototypes. The language provides for type (object) definitions including inheritance and for method prototyping.
- The Unified Modeling Language [19]: a language for software system modeling that contains a series of notations for defining many facets of an object-based system, including the relationships between objects.
- The Internet Inter-ORB Protocol [2]: the backbone of the OMG specifications provides mechanisms for location independent access to services to register, create, and deploy object types and their instances.
- The Object Management Architecture (OMA) [3]: a reference model that describes the framework for the various OMG specifications.
- Existing OMG specifications: specifications include CORBA and IDL, Common Facilities, CORBA Services, and other domain-specific interfaces.
- Bindings of IDL to programming languages: IDL is intended to specify interfaces in a manner that is independent from any particular programming language. The binding of IDL to particular programming languages is the realization of the interface as function calls that are usable by a specific application.

OMG specifications are required to represent interfaces in IDL, which defines only the signatures (names, return types, parameters, thrown exceptions, etc.) of operations and attributes. They also may not reproduce functionality provided in other existing OMG specifications and must use the existing specification if the same functionality is needed. Notably, UML is not required for documenting OMG interfaces; however, it is frequently used. Domain specifications are independent of the programming language bindings. An application uses a domain specification manifest as calls in a programming language. The calls are algorithmically derived by applying the mapping of IDL to a programming language to the interfaces contained in the domain specification.

2.2 OMG and portability

The OMG series of specifications provide run-time portability for applications. This has been demonstrated in the context of a single ORB implementation.[3] Compile time portability is more readily achieved across ORBs. The issues in achieving run-time cross-ORB portability include performance concerns combined with the availability of services on a given ORB.

[3] Run-time portability should not be confused with run-time interoperability. The latter addresses the ability of an application to access services residing on a different ORB at run-time. This capability is largely available using OMG's Internet Inter-ORB Protocol (IIOP) specification.

The OMG standards also support design portability. IDL allows application interfaces to be accessed in a number of programming languages although they only need be implemented in a single language. Additionally, by specifying an operational framework for software specification including the definition (although abstract) of many services, the OMG specifications support design portability for applications.

2.3 STEP

Application interfaces in STEP are based on the following STEP-defined methods:

- EXPRESS [5]: a language for describing data types, constraints on data types, and relationships between data types. The language includes inheritance, function definition, rule definition, domain and relationship constraint definition, but not method prototyping.
- EXPRESS-G: a diagramming technique supporting a subset of the EXPRESS language.
- Existing Integrated Resource Models and Application Interpreted Constructs: two series of information descriptions which cover specific engineering disciplines and other commonly used constructs.
- The Application Protocol (AP) methodology: a specification of the information required for the specification of an AP. Includes the use of process modeling language (IDEF0) to illustrate the scope of a specification and mapping tables to illustrate the details of the relationship of the AP to other information descriptions within the standard.
- The Standard Data Access Interface (SDAI): a specification of a programming interface to data described using EXPRESS, such as is found in other parts of STEP. SDAI defines mechanisms to create, locate, and access sets of data, as well as individual data instances.
- SDAI programming language bindings: provide a realization of the STEP definitions as function calls that are usable for a specific application.

Documentation requirements for STEP specifications are somewhat more rigorous than for OMG specifications. APs, the information models implemented for STEP, are required to contain a representation of the information in both EXPRESS and EXPRESS-G. As with OMG, STEP specifications are not permitted to reproduce functionality that is available in other parts of the standard. A new specification is required to go through a process referred to as "integration" so that the specification will be consistent with other parts of the standard and to ensure that redundant interfaces are not defined.

2.4 STEP and portability

STEP supports system integration where a primary stumbling block to the integration is data representation, in particular, product model data. The mapping of product data among various representations is a difficult aspect of the integration problem. Consequently a primary component of STEP is information models which define a neutral format for data representation. These definitions are a step towards achieving design portability.

In the context of data sharing "design portability" refers to the design of data translators – the modules that manufacturing software vendors implement to translate between their proprietary data representations and the normative representations of STEP. Using a STEP AP a single translator is needed to map data between a component system and the STEP format. The critical part of the design is not the syntactic translation, which is a "simple matter of programming," but the conceptual mapping, where the world view of the vendor is integrated with the world view of STEP. That translator can be coded in as many languages as is necessary but the mapping only needs to be done once. The design is preserved.

STEP defines a selection of implementation formats to support greater degrees of portability. The implementation format available in the initial release of the standard is an ASCII file representation [6]. The file format only supports design portability since it does not include a programmatic interface to the files. Additionally, it only supports portability of the data mapping part of the design for the software that uses it. SDAI supports a greater degree of design portability by specifying methods for accessing and managing sets of data in addition to the individual data values, thereby allowing applications to use a uniform approach for these functions as well. The programming language bindings for SDAI – C++, C, and IDL – support compile time, link time, and run time portability respectively. SDAI does not address plug-and-play capabilities as defined in this paper.

3. Prototype test system: a PDM example

As a whole these standards provide a frame of reference in which open systems can be designed, developed, and maintained. STEP provides a reference for semantics for information exchange and interface standards. OMG provides a broader compatible set of computational interface standards. Operational systems can be developed against those standards utilizing pieces of them as best fit the system requirements. Components of the systems can be tested against the standards but the testing infrastructure should be flexible so as to accommodate the variety of levels of use of the standards. Ideally, tests would be defined such that they can be applied at all stages from system design through run-time operation. Furthermore, the standards' methods provide a framework against which tests can be modularly organized so that different aspects of the system can be tested independently and at different levels of conformance.

Domain specifications in the area of product data management (PDM) are available from both OMG and STEP. The OMG specification borrows heavily from STEP in the definition of data. We are using these PDM specifications to experiment with creating a flexible test system that could be tailored to address the particular profile of the system under test. For a detailed discussion of how the STEP and OMG standards for PDM can be used both individually and in combination see *STEP and OMG Product Data Management Specifications: A Guide for Decision Makers* [17]. If successful the approach taken to aligning these standards will set a trend for combining the exchange-based systems integration approach with the interaction-driven approach. Already, further areas of overlapping concern between the two standards communities have been identified.

3.1 Overview of PDM-related specifications

The PDM Enablers specification makes use of IDL and UML Class Diagrams. The specification has been scoped to exclude significant functions that are often provided by PDM systems but which fall within the scope of other specifications in OMG, such as Workflow. Additionally, the specification includes a description of scenarios for how the specification is to be used. The specification contains twelve IDL modules which support eight separate PDM functions.

The interfaces defined in the PDM Enablers specification provide a combination of data model and data access mechanisms. However, the behaviors exhibited and constraints enforced by conforming PDM implementations are not formally defined. Although there are UML diagrams and much explanatory text, only the IDL interfaces are considered normative.

STEP's AP 203 [8] describes the data necessary for configuration management, which is a large portion of product data management. The PDM Enablers reference AP 203 where possible. In a follow-on project two industrial consortia, PDES Inc. and ProSTEP, have extended AP 203 to cover a broader range of the data used and managed by product data management systems. The result is known as the PDM Schema [16].

The PDM Schema identifies a consistent set of data types to be used in STEP APs, the implementable parts of STEP. As such the PDM Schema contains data definitions described using the EXPRESS language via references to other parts of STEP, such as the Product Structure Schema contained in the integrated resources. Data corresponding to the PDM Schema consists of a subset of the data needed to exchange information for a particular AP. (For example, the product structure and configuration portion of the data from an exchange using AP 203, which also includes a three-dimensional representation of the product, would satisfy the PDM Schema.)

The application of SDAI to the PDM Schema results in a set of standardized application interfaces for PDM Systems. Using SDAI the EXPRESS representation of the PDM Schema is rendered as a series of callable interfaces in both C and C++ and also an abstract interface in IDL, which in turn produces callable interfaces in a number of programming languages. We refer to these API as a SDAI/PDM interface.

Both the PDM Enablers and the PDM Schema interfaces allow applications to access PDM systems using a standardized interface. Both support access by multiple programming languages. As with other STEP parts the focus of the PDM Schema is on the data needed for exchange between systems in order for those systems to interoperate. In contrast the OMG PDM Enablers specification focuses on the operations needed to manage product data, such that other types of systems may use the PDM.

The addition of a functional specification to the PDM data narrows the range of permissible operations on the data by imposing an order on them. For instance, in the PDM Enablers specification many relationships cannot be created independently from instances that they are relating and they cannot be changed once they are created. With the PDM

Schema a change to a relationship is simply a matter of changing a field of a relationship entity to point to a different entity. This capability represents a change in state of the PDM data and may result in corrupt data. The interfaces defined in the PDM Enablers specification act as guards to prevent this sort of corruption. For example, with the PDM Enablers specification a DocumentMaster object is needed to create a DocumentVersion object. Once that DocumentVersion object has been created it cannot be "reassigned" to a different DocumentMaster. Such an operation would be logically incorrect and could result in bad data if another object pointed to the document version with the assumption that it related to a particular document master. SDAI does not address these types of restrictions on what changes can be made to the state of the data. In addition, an automatic way to inject such restrictions is not obvious since the restrictions address the semantics of change of data relationships. The constraints specified for STEP are limited to the semantics of the static relationships. For example, the STEP models constrain that a "version" must be a version *of* something (a "master"), but they do not capture the intent that this relationship should be immutable, i.e., that it makes no sense for a "version" to be linked to a different "master" than it was originally.

3.2 System design

Three necessary components for an abstract test suite are usage scenarios, data, and test criteria. The prototype test system uses usage scenarios (describing the operations to be performed) that are documented as part of the PDM Enablers specification and PDM data which is available in the STEP format using the PDM Schema. (The STEP APs also contain usage scenarios; however, all the existing scenarios are for data exchange and not interactive data sharing.) Combining these two pieces results in 2/3rds of an abstract test suite for interactive data sharing. In the absence of formal test criteria we have chosen to support the comparison approach to testing rather than try to impose hard criteria and test exhaustively. The comparison approach compares the results of two implementations to determine whether they match. This approach also seemed to fit the intent of the standards, that is to support system interoperability by wrapping existing systems.

Figure 1 illustrates how STEP-based PDM data can be used to drive the generation of executable tests. This approach has the additional advantage that it can be used to validate the ability of either a PDM Enablers implementation or an SDAI/PDM implementation to handle existing data that was produced for file exchange. The shortcoming of this approach is that it is limited to a usage scenario based on file exchange and does not involve any of the more fine-grained operations that are available using the PDM Enablers or an SDAI/PDM interface. However, in our prototype test system we selectively use portions of the generated client code to flesh out the detailed data content of the PDM Enablers interaction-driven scenario. This step was not fully automated but it did allow us to take advantage of the rich data that was available for file exchange and apply it in a different context. Developing such data by hand would have been time consuming and error prone.

[Figure 1]

The design of the Mapping/Code generator component of the system is also noteworthy. The EXPRESS Engine used in the prototype system is NIST's Expresso toolkit [4]. Expresso supports the EXPRESS-X language [10] which can be used for creating views of data. The semantic part of the mapping of the PDM Schema data into PDM Enablers objects was captured in EXPRESS-X. The Expresso engine then processed the data and generated views corresponding to the PDM Enabler objects. Based on those views, code was then written to generate compilable code that calls the PDM Enablers. This separation of concerns, the semantics of the mapping and the code generation aspect of the task, made development simpler. The mapping semantics could be examined against a data set independently from the code generation. Additionally, the resulting mapping is in an implementation independent format, EXPRESS-X. The EXPRESS-X format is more conducive to review by other parties than had the mapping been embodied in a programming language, yet still a computational format that is capable of being directly processed.

Figure 2 illustrates a complementary approach that uses calls to the PDM Enablers to generate SDAI calls that mirror those operations. Although the PDM Enablers calls, i.e. the test scripts, would need to be hand-generated (but partially automatable using the previous approach), the system could automatically generate tests based on realistic scenarios for a SDAI/PDM implementation.

[Figure 2]

These two systems combined can provide a large number of reasonable executable tests. Our prototype scenarios defined in the PDM Enablers specification are fleshed out using data available from STEP PDM Schema testing activities.

The final piece of the test system is the Comparator, which is used in determining the test verdicts. The Comparator code is linked into the executable tests to enable the results of two runs of the test against two different PDMs to be compared for consistency. The comparison is not a complete evaluation of whether a system supports the interfaces. Instead, inconsistencies, identified by the comparison, flag potential errors that can then be analyzed to identify the source of the error.

4. Conclusion

In summary, we have designed a test system that when configured one way can be used to generate tests of a PDM Enablers implementation and when configured differently can generate logically equivalent tests for an SDAI implementation. Additionally, scenarios and data defined for each of the standards may be used in the context of the other standard, where overlap in the content exists. The methods employed in defining these standards are essential to enabling this flexibility.

Our test system architecture is designed modularly so that it can accommodate changes in the configuration of the system under test. It can be modified to support the generation of test clients in different interface styles and in multiple programming languages. Furthermore, the approach can be duplicated to support interfaces in areas other than PDM. While the code generation software would need to be rewritten, the tools on which it is based could be reused. OMG is currently seeking proposals for object interfaces to CAD models [13]. Such an interface would have overlaps with STEP similar to those in the PDM area, so our approach to test case development will be reusable.

References

[1] CORBA 2.3.1 Chapter 3 (IDL Syntax and Semantics), http://cgi.omg.org/cgi-bin/doc?formal/99-07-07 (1999).
[2] CORBA 2.3.1 Chapter 15 (General Inter-ORB Protocol), http://cgi.omg.org/cgi-bin/doc?formal/99-10-11 (1999).
[3] Discussion of the Object Management Architecture (OMA) Guide, http://cgi.omg.org/cgi-bin/doc?formal/00-06-41 (2000).
[4] Expresso Homepage, http://www.nist.gov/expresso (1999).
[5] ISO 10303-11:1994: Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual (1994). Available from ISO, http://www.iso.ch/.
[6] ISO 10303-21:1994: Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure (1994). Available from ISO, http://www.iso.ch/.
[7] ISO/DIS 10303-22:1996: Industrial automation systems and integration — Product data representation and exchange — Part 22: Implementation methods: Standard data access interface (1996). Available from ISO, http://www.iso.ch/.
[8] ISO/DIS 10303-203:1994, Industrial automation systems and integration — Product data representation and exchange — Part 203: Configuration Controlled 3D Designs of Mechanical Parts and Assemblies (1994). Available from ISO, http://www.iso.ch/.
[9] ISO/IEC 10746-1:1998, Information technology — Open Distributed Processing — Reference model: Overview (1998). Available from ISO, http://www.iso.ch/.
[10] ISO TC184/SC4/WG11, EXPRESS-X Language Reference Manual, http://www.steptools.com/library/express-x (1999).
[11] S. J. Kemmerer, ed., STEP the Grand Experience (National Institute of Standards and Technology, Special Publication 939, 1999). Available at http://www.mel.nist.gov/msidlibrary/summary/9920.html.
[12] Manufacturing Domain Task Force home page, http://www.omg.org/homepages/mfg/ (2000).
[13] Manufacturing Domain Task Force Roadmap Version 4.0, http://www.omg.org/cgi-bin/doc?mfg/99-09-15 (1999).
[14] Object Management Group home page, http://www.omg.org/ (2000).
[15] PDM Enablers v1.3 Publication draft, http://cgi.omg.org/cgi-bin/doc?mfg/00-07-02 (2000).
[16] PDM Schema, v1.1 is a pre-normative specification available from http://www.pdm-if.org/ (1999).
[17] D. Starzyk, STEP and OMG Product Data Management Specifications: A Guide for Decision Makers, http://cgi.omg.org/cgi-bin/doc?mfg/99-08-02 (1999).
[18] Technical Committee 184: Industrial Automation Systems and Integration Subcommittee 4: Industrial Data, http://www.nist.gov/sc4/ (2000).

[19] Unified Modeling Language (UML) 1.3 specification, http://cgi.omg.org/cgi-bin/doc?formal/2000-03-01 (2000).

KC Morris is a computer scientist in the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology (NIST). She came to NIST after receiving a Master of Science degree in Computer Science from the Illinois Institute of Technology in 1988. KC has been a primary contributor to the development and validation of the implementation mechanisms of ISO 10303, the Standard for the Exchange of Product Model Data (STEP). She contributed to the design and development of numerous prototypes of the STEP Data Access Interface (SDAI) including portions of NIST's public-domain STEP Toolset (http://www.nist.gov/scl). KC has published dozens of articles on these topics and has actively worked with the ISO technical working groups developing STEP and the industrial consortia PDES Inc. and NIIIP. Her current research interests include object-oriented and distributed systems for engineering and manufacturing, product data management, and techniques for testing such systems.

David Flater is a computer scientist in the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology (NIST). He came to NIST in 1992 while completing his Ph.D. in Computer Science, which he received from the University of Maryland in 1994. From 1992 to 1996 he was a primary contributor to the NIST SQL conformance test suite in the Computer Systems Laboratory. In 1996 he transferred to the Manufacturing Engineering Laboratory where the focus shifted from SQL to the Common Object Request Broker Architecture (CORBA). He contributed to a number of projects involving CORBA and developed the Manufacturer's CORBA Interface Testing Toolkit (http://www.mel.nist.gov/msidstaff/flater/mcitt/). His current research interests include object- and agent-oriented systems for engineering and manufacturing, testability, and standards quality.

Table 1: Interface specification methods

Group	OMG	STEP
Definition language	IDL	EXPRESS
Diagramming technique	UML's Class Diagrams	EXPRESS-G
Architectural framework	Object Management Architecture	AP implementation methodology
Operational framework	IIOP	SDAI
Modeling language	UML	EXPRESS
Complimentary specifications	Common Facilities, Common Services, other domain specifications	Integrated Resource Models, Application Interpreted Constructs, Application Modules
Programming Language Support	IDL Language Bindings	SDAI Language Bindings

Figure 1: STEP data-driven approach to test generation

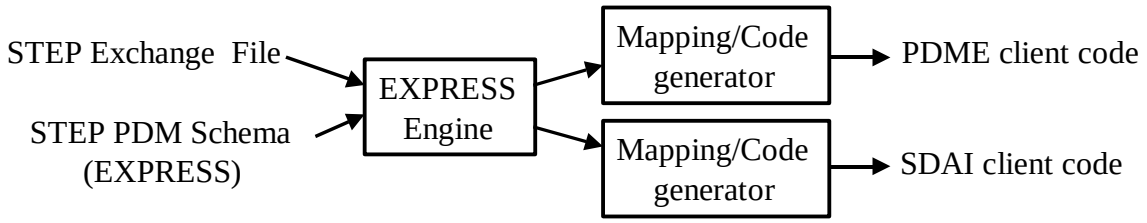

Figure 2: Scenario-driven approach to test generation