The NIST EXPRESS Toolkit

—

Requirements for Improvement

Don Libes
James Fowler

Abstract

The NIST EXPRESS toolkit is a software library for building EXPRESS-related tools. This paper is one in a series describing the latest version of the toolkit. This document describes shortcomings of previous versions of the toolkit, requirements for improvement, and a recommended approach for addressing those requirements. A background knowledge of EXPRESS and the EXPRESS toolkit is presumed as well as a rudimentary grasp of basic compiler construction techniques.

Introduction

The NIST EXPRESS toolkit [1] is a software library for building software tools for manipulating information models[1] written in the EXPRESS language [2]. An example application is included which reports syntactic and semantic errors in EXPRESS schemas [3]. Work on the toolkit has been funded by the Department of Defense's Computer-aided Acquisition and Logistic Support (CALS) program[2]. The toolkit was first made available in 1990 and has been periodically updated to keep pace with changes in the EXPRESS language and to correct errors in its operation. The toolkit is freely available and used widely in private industry, government facilities, and universities.

In April 1992, a version of the toolkit was released which supported the *CD Ballot version* (often referred to as the "N14 version" because of the document's number in the ISO numbering scheme) of the EXPRESS language[3]. During the development of that version of the toolkit (known as the "N14 toolkit release"), it became increasingly difficult to retrofit corrections and additions to the fundamental implementation of the toolkit. After April, additional changes to the EXPRESS language resulted from the CD Ballot process. In order to support the latest changes to EXPRESS, the impact on the implementation of the toolkit had to first be addressed. A revision to the

---

1. The terms *information model*, *data model*, and *conceptual schema* are used interchangeably throughout this document.
2. The EXPRESS toolkit and documentation are works of the U.S. Government and therefore not subject to copyright.
3. The term *CD Ballot version* refers to the fact that the EXPRESS language reference manual had been accepted by a select set of member countries in the International Organization for Standardization (ISO) as ready to be submitted to a larger vote for international standardization. The term *DIS* (Draft International Standard) signifies that a document is undergoing the wider voting process which can make it an international standard. EXPRESS is currently a DIS.

fundamental design of the toolkit was considered in response to the growing list of requirements for the toolkit's functionality - requirements which could not be readily addressed with the existing implementation. This document describes those requirements and their implications on the design of the toolkit.

On the basis of this document, an entirely new EXPRESS toolkit is in the process of being designed and implemented. We anticipate that the new toolkit will meet all of the requirements listed here, subject to the continuing underlying requirements of performance, flexibility, conformance, and functionality as a research testbed.

Requirements

The requirements for improvement to the toolkit fall into three basic categories (although there is some overlap between these):

A. Changes required to support the DIS specification of EXPRESS.

B. Changes helpful to improving the efficiency of the toolkit and software derived from it.

C. Changes helpful to or requested by users but not otherwise covered by (A) or (B).

A. Support for the EXPRESS DIS

A.1 *Support the ALIAS construct.*
The ALIAS construct was introduced in the N14 version of EXPRESS but is not supported in the N14 toolkit release. Using the current design of the toolkit, this concept requires that the implementation of the toolkit support multiple inheritance.[4] Unfortunately, multiple inheritance will severely degrade performance. The toolkit should be redesigned to avoid this degradation or to handle this concept without multiple inheritance.

A.2 *Support the RULE construct.*
The N14 toolkit release does not support the RULE construct entirely; RULEs are parsed but not resolved. RULE resolution is complicated by scoping considerations which are more easily addressed with improvements to the original toolkit design.

A.3 *Support the QUERY construct.*
As with the RULE construct, QUERYs are parsed but not resolved. Handling QUERYs involves the same kinds of considerations for change to the toolkit as do RULEs.

A.4 *Handle the USE/REFERENCE construct correctly.*
Aside from being error-prone, the N14 toolkit release handling of USE/REFERENCE is extremely inefficient. Items referenced through more than one USE/REFERENCE require significant memory since deep dictionary copies are made. Certain cases (including simple user errors) lead to infinite recursion.

4. The difficulties of implementing these and other constructs (noted elsewhere in this paper) using single-inheritance as implemented in this toolkit have been described already [5].

A.5 *Handle Enumerations correctly.*
The N14 toolkit release handles Enumerations such that Enumerations in the same scope place all items in that scope. Thus it is impossible to have similarly-named items in different enumerations. The obvious way to correct the flaw in the current implementation is to use multiple inheritance. Again, employing multiple inheritance will severely impact performance. If only as an excuse, we note that details of enumeration scoping have changed in every recent draft of EXPRESS.

A.6 *Support the GENERIC tag construct.*
The GENERIC tag was added in the N14 version of EXPRESS and is not supported in the N14 toolkit release. Supporting the construct using the original toolkit design appears to be quite difficult to implement in the existing type hierarchy.

A.7 *Handle implicitly-defined loop control variables correctly.*
Loop control variables which occur in an EXPRESS infomation model have been a persistent source of problems in all versions of the toolkit. The problems with handling the implicitly-defined variables in the toolkit stem from the inheritance hierarchy. Fixing the problem using the existing inheritance model will degrade performance.

A.8 *Support locally-defined CONSTANTs.*
Locally defined CONSTANTs have never been implemented at all in previous releases of the toolkit. Support for the feature require new data type handling and some changes to the existing scope searching mechanism.

A.9 *Fix errors in interpretation by the EXPRESS parser.*
A number of minor errors exist in the N14 toolkit release. For example, the GENERIC construct is not handled per the interpretation prescribed in the DIS version.

B. Reduced complexity and higher efficiency

B.1 *Improve overall performance.*
The object-oriented nature of the N14 toolkit release incurs a severe toll on efficiency. All structure elements are accessed by function call, resulting in geometric time complexity. It should not be necessary to access all elements through function calls.

B.2 *Reduce code complexity.*
In the N14 toolkit release, symbols are resolved as they are encountered. This approach has several drawbacks. (1) Stacks can grow arbitrarily deep. It is not uncommon to find stacks that are hundreds of calls deep. This makes debugging overwhelming, and suggests that it would be impossible to port the implementation to small machines with limited stack space. (2) Objects have to be “self-aware.” Since objects are not visited in a systematic way, objects must provide their own knowledge of whether they should be resolved. This is inefficient, and in the end analysis, an unnecessarily complex approach.

B.3 *Improve memory management.*
The N14 toolkit release is designed so that most access functions copy objects. This is clean in theory, but not in practice, since few toolkit applications destroy objects or make arrangements to do so. NISTIR 4814 stated, “memory management ... is not given high priority in the current implementation” [4]; closer attention should be paid to memory management.

B.4 *Remove duplicate string functions.*

The string abstraction in the N14 toolkit release duplicates some of the Standard C Library. The Standard C functions are tuned to each implementation, and can usually be expected to be much more efficient than what is provided in the toolkit.

B.5 *Improve Diagnostic reporting.*

The N14 toolkit release takes significant time just to report the very simplest of errors. The cause is that two processes are started in order to report diagnostics. This is inappropriate since it is a heavy penalty for simple problems, and, at worst, prevents errors from being reported at all in the case that the system has run out of resources (i.e., out of memory).

Also the system could be more robust in the face of internal errors. For example, the current implementation saves a memory image upon encountering an internal error. This can take a significant amount of time and space. But most users are not interested in the internal details so a memory image could be saved only if the user specifically requested it. This would save both space and time upon processing internal errors.

C. User Support

C.1 *Improve support for application development.*

The N14 toolkit release names the basic function calls after the pass numbers. The application programmer thus has to know the pass numbers and be aware of what happens during each. This could be ameliorated through a different calling paradigm.

The N14 toolkit release also provides abstractions to access objects, but provides no abstractions to actually manipulate the objects themselves. For example, ENTITYget_attributes is a function call which hides the actual details of obtaining the attributes of an entity. But the user must recognize that the attributes are returned as a list in order to make use of them. This should be remedied to avoid recoding of applications as the toolkit changes.

The N14 toolkit release provides a sample main function. The main function provided assumes a strict idea of EXPRESS applications. Instead, it should be more flexible in order to accommodate a wider range of toolkit applications.

An EXPRESS-application main function should provide:

- Options to manipulate diagnostic buffering.
- Options to skip resolution.
- Options to print run-time updates based on object types.
- Provision for reporting a version.
- Provision for providing optional application-specific: versions, backend processing, initialization, and option processing.

In addition, a dynamic-linking option provided in the implementation has not been used by application developers and there is no reason to expect it will be. It could be discarded, and the code space reclaimed.

C.2 *Improve Error messages.*

The parser in the N14 toolkit release needs to report better error messages. It indicates what token is found in a syntax error - but it should also describe what tokens are expected, and what scope the error is found in.

Many resolution-time error messages are overloaded (e.g., “syntax error”), obscuring what the errors really are.

C.3 *Provide file-to-schema translation mechanism.*

The INCLUDE keyword was phased out in N14, leaving the details of file-to-schema translation up to the implementation. (The EXPRESS committee recognized their own lack of knowledge in this area.) The INCLUDE concept is redundant since the USE/ REFERENCE construct is sufficient to reference external schemas.

Schemas mentioned by USE/REFERENCE but not otherwise found in the current file should be incorporated by an implicit but otherwise surreptitious mechanism. This mechanism should understand directory paths and libraries of files of other schemas. External files should be able to contain multiple schemas.

Reincorporating multi-file handling as these changes suggest requires that the error message handler understand file names (in order to report them to users). This will require changes to the symbol and construct abstraction. In the current implementation, this is yet another example where multiple inheritance would be useful.

C.4 *Port to PC; Recode for Standard C*

Attempts were made to port the N14 toolkit release to a personal computer class platform. These failed for reasons unknown[5]. Data and stack space consumption are obvious candidates. The original implementation used a number of non-conforming implementation-defined techniques, any one of which could be problematic in some computer environments. This non-portable code should be fixed.

C.5 *Define run-time environment*

EXPRESS alludes to but does not define a run-time environment. Defining a run-time environment would be helpful in error checking and use of the toolkit in other applications (e.g., STEP data exchange file processing).

C.6 *Support lint-like behavior*

The N14 toolkit release performs a limited amount of error checking during parsing and translation. The philosophy is to not go out of the way to detect errors – simply to report them if easily detected.

A lint-like[6] program or option could be used to indicate that the user is interested in a more extensive analysis without regard to time. That is, given more time, more extensive error-checking will be performed.

---

5. These attempts were reported to the toolkit developers but without enough information to specifically respond.
6. Lint is a standard Unix utility which attempts to detect features of C programs that are syntactically correct but undesireable for reasons of inefficiency or non-portability.

Conclusions

As was mentioned earlier, addressing several requirements implies the implementation of multiple inheritance. The current toolkit is implemented around an object-oriented engine written in the C programming language. It is inconceivable that employing multiple inheritance in this environment will not slow the toolkit's performance. In fact, multiple inheritance would be very expensive computationally (since the toolkit's object-oriented manipulators themselves are interpreted at run-time rather than being compiled).

The object-oriented implementation used in the N14 toolkit release and its predecessors can no longer support the functionality demanded by the latest version of EXPRESS without sacrificing execution speed. The original design was worthwhile for building a prototype, but experience has progressed the developers beyond the prototype stage. In addition, users have grown more sophisticated with the use of the toolkit over time, leading to requests for additions and improvements which are best handled with a new implementation design.

Recommendations

It should be possible to mix the best elements of the original object-oriented paradigm (OOP) with those of a more classic compiler (non-OOP) approach. In particular, the original object hierarchy could be compressed from five levels to two or three. It is expected that certain sections of code (namely, types and expressions) will be full of case statements. It may be possible to ameliorate just these sections with judicious use of object-style tables and unions.

Instead of the current single-pass resolver, a multiple-pass resolver should be used. This willgreatly reduce the stack depth required, and alleviate the complexity required by having to encounter objects that are in various stages of resolution. Such an approach will form the basis for addressing all of the requirements described in this document.

References

[1] Clark, S.N., Libes, D., "The NIST PDES Toolkit: Technical Fundamentals," NISTIR 4815, April 1992.

[2] Spiby, P., ed., "ISO 10303 Industrial Automation Systems – Product Data Representation and Exchange – Part 11: Description Methods: The EXPRESS Language Reference Manual", ISO DIS 10303-11:1992(E), July 15, 1992.

[3] Clark, S.N., Libes, D., "FED-X: The NIST EXPRESS Translator," NISTIR 4822, April 1992.

[4] Clark, S.N., Libes, D., "NIST EXPRESS Working Form Programmer's Reference," NISTIR 4814, April 1992.

[5] Libes, D., Clark S.N., "The NIST EXPRESS Toolkit - Lessons Learned", *Proceedings of the 1992 EXPRESS Users' Group (EUG '92) Conference*, Dallas, TX, October 17-18, 1992.