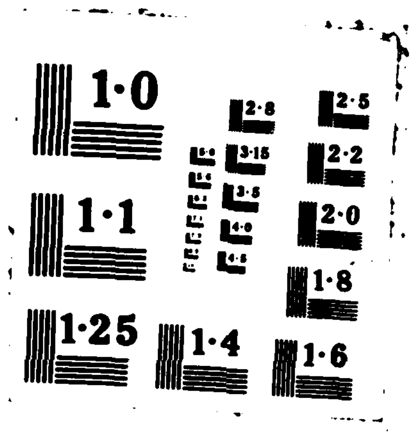
AD-A188 922 IDL (INTERFACE DESCRIPTION LANGUAGE) BACKGROUND AND STATUS(U) CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.. D L STONE ET AL. DEC 87 1/1

UNCLASSIFIED CMU/SEI-87-TR-47 ESD-TR-87-210 F/G 12/5 NL


1·0
2·8
2·5
3·15
2·2
3·5
1·1
4·0
2·0
4·5
1·8
1·25
1·4
1·6

(2)

Technical Report
CMU/SEI-87-TR-47
ESD-TR-87-210

Carnegie-Mellon University
Software Engineering Institute

# IDL: Background and Status

Donald L. Stone
John R. Nestor

December 1987

**Technical Report**
**CMU/SEI-87-TR-47**
**ESD-TR-87-210**
**December 1987**

# IDL: Background and Status

**Donald L. Stone**
**John Nestor**

**Software Engineering Institute**
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

**Review and Approval**

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]
Karl Shingler
SEI Joint Program Office

This work was sponsored by the U.S. Department of Defense.

# IDL: Background and Status

**Abstract.** This paper presents an overview of the Interface Description Language (IDL). We describe the language and its history. We also discuss the status of the IDL community.

## 1. What is IDL?

Programming environments contain sets of tools that communicate with one another. Often the shared data is highly structured and extremely complex. The Interface Description Language (IDL) provides a mechanism by which the abstract properties of complex structured data may be specified precisely. Abstract specifications can be mapped to multiple target languages running on many kinds of computer systems. This mechanism permits data to be communicated between programs, or parts of a single program, safely and efficiently.

The range of existing tool communication methods can be illustrated by two examples. First, one may specify a single central database shared by all the tools in the environment. This approach, use of a single powerful data representation, is that taken by the Gandalf programming environment [4]. Alternately, one may specify a particular data structure as the interface between each pair of tools. By providing a powerful mechanism for specifying and generating data structures, IDL enables the system designer to tailor tool interfaces to specific needs.

Programs in which interfaces are specified in IDL communicate by means of reader/writer pairs. An interface consists of a reader, which gets data, a writer, which sends data, and some exchange representation. A writer transforms data in the internal form of its process to the exchange representation. A reader then transforms this representation into the internal form of its own process.

IDL supports a common exchange representation, the ASCII External Form. This form can describe data from any IDL specification independent of language or machine, but it is often inefficient. However, in practice the ASCII form serves as a lowest common denominator representation. An IDL translator can also support more efficient binary exchange forms.

The core of IDL is a language for specifying the abstract properties of data structures. The structures specified are typed, attributed, directed graphs. Each node in the graph has a collection of *attributes*. Attributes can have *types*: integer, rational, boolean, string, set, sequence, private or node. *Private* types are a hook allowing the tool builder to specify arbitrary structures. The nodes form a graph because attributes may reference other nodes.

IDL defines the notion of a *class*. A class is a grouping of nodes that share a set of common attributes. Operations that reference only these common attributes may be defined on classes. The class mechanism can be used to add a kind of generic capability to target languages lacking such capability. Also, classes may be further grouped, forming a class structure. An extremely important property of this structure is that it need not be hierarchical.

An IDL translator maps an abstract IDL specification to a concrete specification in some target language. In building a translator, a number of issues are left to the discretion of the implementor. Among these:

- How does the IDL type model map onto the target language?
- How does the application programmer manipulate objects within the IDL type model?
- How much runtime support is generated?

In addressing these issues, an IDL implementor can achieve tremendous leverage. Since it need never be read by humans, generated code may be written in an unreadable and unmodifiable style, in order to be as efficient and effective as possible. Furthermore, generated code may be arbitrarily machine and language specific.

An IDL tool presents another benefit to the application programmer. In general, interfaces to data structures and runtime support constitute a sizable portion of systems. An IDL tool will generate this code correctly. Furthermore, the runtime support may use highly-tuned tools. For instance, beneath the type manager, one could put a highly-tuned storage manager and garbage collector, which then would be incorporated into every system built with IDL. When an entire toolset is based on IDL, these secondary benefits can be significant.

In addition to the language for describing data structures, IDL contains several other concepts. Among these is a language for specifying *assertions*. This language allows us to make statements about data structures such as "all the back pointers in a data structure are set correctly." Another piece of IDL, the process model, allows us to make statements about how processes interact, such as "they communicate in main memory" or "they pass some external representation through files." Assertions and process statements give us more information to reason about systems and to build tools to support interactions among pieces of systems.

Finally, IDL is formally defined in denotational semantics. In the past, this has had a very pragmatic effect. Proposed changes and implementation decisions were weighed against the requirements of the formal definition. In the long run, this has led to a cleaner, more coherent language.

## 2. History of IDL

The Production Quality Compiler-Compiler (PQCC) project at Carnegie-Mellon University (CMU) [6] developed a notation called LG (for *Linear Graph*) to describe the data structures passed between phases of the compiler [10]. The project also developed a set of tools for processing the notation. However, LG had a number of drawbacks. It was difficult to use, and it was strongly oriented towards the particular implementation language (BLISS [13]) and host machine (the PDP-10) used by the PQCC project. Nonetheless, it was a very useful tool.

At CMU, during 1979 and 1980, a consensus developed that a generalized data definition language was needed to meet the needs of several different projects, which were implemented in different languages and running on different computer systems. During this same period, the community of implementors of the Ada programming language developed a strong interest in being able to share intermediate program representations.

In late 1980, there were two major candidates for a common intermediate representation of Ada programs: TCOL$_{Ada}$, developed at Carnegie-Mellon University, and AIDA, developed at the University of Karlsruhe. In December 1980, a meeting was held at SofTech, Inc., to discuss these two representations; at this meeting, it was decided to try and merge the two notations. In January 1981, a one-week design session was held at Eglin Air Force Base. The result of this meeting was a new intermediate representation, Diana [3].

As a shared representation, Diana needed precise specification. Previously, John Nestor, David Lamb, and William Wulf had defined an initial design of IDL. At the Eglin meeting, the participants revised this design and used it as Diana's specification language.

Soon afterward, the IDL reference manual [7] appeared, containing the first version of the formal definition. This was followed by a draft revision [8] containing only minor changes. At that time, the first implementation of an IDL translator was built at CMU. Later, David Lamb's PhD thesis [5] investigated the practicality of using IDL to build large systems and explored many of the design issues.

In 1981, Wulf and Nestor founded Tartan Laboratories, using IDL as a central technical basis for their compiler building tools. Tartan has continued to support work on IDL, including a revision of the formal definition [1], work on extensions to attribute grammars using IDL [9], and a runtime system built by Joseph Newcomer.

Several other sites have implemented IDL translators. Intermetrics has two implementations supporting their Ada development effort. A number of other companies building Ada compilers have adopted Diana and built partial translators for support. More recently, a Diana-like language called Ivan, also specified in IDL, has appeared as part of the VHDL system [2].

Currently, the University of North Carolina at Chapel Hill (UNC) is using IDL as the basis of an integrated toolset [11]. To support this effort, they are implementing an IDL translator targeted to C, Pascal, Modula-2 and Ada. They are the first group to work extensively with parts of IDL other than the data structure description language. They have worked with both the assertion language and the process model. Their work using the whole of IDL is driving many of the changes and extensions currently under consideration.

In January 1986, the Software Engineering Institute of Carnegie-Mellon University (hereafter the SEI) initiated a project on IDL as part of a larger focus on manipulation of data in programming environments. The major product of the project will be a book containing

- A revised language reference manual incorporating changes and extensions to IDL.
- A guide to implementors describing experiences and containing notes about implementing IDL efficiently and effectively.

In May 1986, the SEI and the Computer Science Department of UNC cosponsored a workshop on IDL. The workshop brought together people who have been significant contributors to IDL and IDL-like technologies. The objective was to exchange ideas and open lines of communication for future work.

## 3. Status of IDL

In preparation for the IDL workshop, the SEI project surveyed a number of users and implementors of IDL and IDL-like systems. We talked to fifty people and gathered names of about one hundred more. From them, we identified thirteen implementations of IDL and ten implementations of IDL-like systems. This work convinced us that these technologies are gaining acceptance in a broader section of the community.

We see three issues facing the IDL community today.

- To make IDL more accessible.
- To consider a number of language changes and extensions.
- To collect and disseminate strategies and techniques for implementing IDL.

Until recently, the only available introduction to the concepts of IDL was the language reference manual. As a part of their system, UNC has written a tutorial [12] that provides a far more coherent introduction. In addition, the general availability of a tool (the UNC system) should aid novices in gaining a full appreciation of the technology.

A number of people currently are working on revisions to IDL. After some years of experience with the ideas, many changes and extensions to the language have been proposed. Most of these are in areas outside the core language; for example, a completely revised process model has been proposed. A new reference manual, to be published as part of the SEI book, will incorporate some new design work based on these proposals.

The IDL workshop provided a forum in which to discuss proposed language changes. Many of the participants presented new work, and the group provided a great deal of good technical feedback. As work on the new design continues, a new Arpanet mailing list, Info-IDL (see announcement) will become the forum for this discussion.

A major reason IDL has not been used widely in the past is that strategies for implementing such tools efficiently are not obvious. Experience has shown that there are a number of practical engineering "tricks" that are extremely useful in building an IDL system. Discussions with the participants at the workshop confirmed the belief that other implementors have discovered their own sets of tricks. The implementor's guide in the SEI book will present many of these experiences, tricks and strategies in a framework that will aid future implementors.

## References

[1] Paola Giannini.
A Denotational Semantics for IDL.
1986
To be published as a Technical Report of the Computer Science Department, Carnegie-Mellon University.

[2] Alfred S. Gilman.
VHDL - The Designer Environment.
*IEEE Design & Test 3*, April, 1986.

[3] G. Goos and W. A. Wulf (editors).
*Diana Reference Manual.*
Technical Report CMU-CS-81-101, Carnegie Mellon University, Computer Science Department, March, 1981.

[4] A. N. Habermann.
The Gandalf Research Project.
*Computer Science Research Review*, 1978-79.

[5] David A. Lamb.
*Sharing Intermediate Representations - The Interface Description Language.*
Technical Report CS-83-129, Computer Science Department, Carnegie-Mellon University, May, 1983.

[6] B. W. Leverett, R. G. G. Cattell, S. O. Hobbs, J. M. Newcomer, A. H. Reiner, B. R. Schatz, W. A. Wulf.
*An Overview of the Production Quality Compiler-Compiler Project.*
Technical Report CMU-CS-79-105, Carnegie Mellon University, Computer Science Department, February, 1979.

[7] J. R. Nestor, W. A. Wulf, D. A. Lamb.
*IDL - Interface Description Language.*
Technical Report CMU-CS-81-139, Carnegie-Mellon University, Computer Science Department, September, 1981.

[8] J. R. Nestor, W. A. Wulf, D. A. Lamb.
IDL - Interface Description Language.
Draft Revision 2 .
1982

[9] John R. Nestor, Bhubaneswar Mishra, William L. Scherlis, William A. Wulf.
*Extensions to Attribute Grammars.*
Technical Report TL 83-36, Tartan Laboratories, April, 1983.

[10] J. M. Newcomer, R. G. G. Cattell, P. N. Hilfinger, S. O. Hobbs, B. W. Leverett, A.H. Reiner, B. R. Schatz, W. A. Wulf.
*PQCC Implementor's Handbook.*
Internal Documentation, Carnegie Mellon University, Computer Science Department, October, 1979.

[11] Richard Snodgrass and Karen Shannon.
*Supporting Flexible and Efficient Tool Integration.*
Softlab Document No. 25, Computer Science Department, University of North Carolina, April, 1986.

[12] W. B. Warren, J. Kickenson, R. Snodgrass.
*A Tutorial Introduction to Using IDL.*
Softlab Document No. 1 Computer Science Department, University of North Carolina, November, 1985.

[13] W. A. Wulf, D. B. Russell, and A. N. Habermann.
BLISS: a Language for Systems Programming.
*Communications of the ACM* 14(12), December, 1971.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A188922

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS NONE |
| 2a. SECURITY CLASSIFICATION AUTHORITY N/A | 3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-87-TR-47 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-87-210 |
| 6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INSTITUTE; 6b. OFFICE SYMBOL (If applicable) SEI | 7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE |
| 6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213 | 7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE, MA 01731 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE; 8b. OFFICE SYMBOL (If applicable) SEI JPO | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962885C0003 |
| 8c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY SOFTWARE ENGINEERING INSTITUTE JPO PITTSBURGH, PA 15213 | 10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: ; PROJECT NO.: N/A; TASK NO.: N/A; WORK UNIT NO.: N/A |
| 11. TITLE (Include Security Classification) IDL: BACKGROUND AND STATUS | |
| 12. PERSONAL AUTHOR(S) DONALD L. STONE, JOHN R. NESTOR | |
| 13a. TYPE OF REPORT FINAL; 13b. TIME COVERED FROM ____ TO ____ | 14. DATE OF REPORT (Yr., Mo., Day) DECEMBER 1987; 15. PAGE COUNT 12 |
| 16. SUPPLEMENTARY NOTATION | |
| 17. COSATI CODES FIELD / GROUP / SUB. GR. | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) IDL, TOOL COMMUNICATION, DATA DESCRIPTION LANGUAGES |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

THIS PAPER PRESENTS AN OVERVIEW OF THE INTERFACE DESCRIPTION LANGUAGE (IDL). WE DESCRIBE THE LANGUAGE AND ITS HISTORY. WE ALSO DISCUSS THE STATUS OF THE IDL COMMUNITY.

| | | |
|---|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED XX SAME AS RPT. ☐ DTIC USERS XX | 21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED | |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL KARL SHINGLER | 22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630 | 22c. OFFICE SYMBOL SEI JPO |

DD FORM 1473, 83 APR    EDITION OF 1 JAN 73 IS OBSOLETE.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE