A UNIVERSAL DELOGGER USING THE ADA SEMANTIC INTERFACE SPECIFICATION (ASIS)

Mickey White

IBM Federal Systems Company (FSC)

Boulder, Colorado

OVERVIEW

Every non-embedded real time system faces the problem of offline analysis of data recorded during real time. In Ada, the mechanism for defining data structures is via 'types'. The Ada language leaves to the compiler vendor the physical implementation of the data structures. This means that an application trying to read back and analyze the data created by the real time Ada system needs to either:

- 'with' all of the types to be analyzed
- or somehow 'know' how the compiler physically mapped the data

In a typical real time system, it is not uncommon for more than a thousand complex data types to be involved in this kind of data recording and analysis. The methods used to date for applications of this kind have included various combinations of:

- 'withing' all of the types into one analysis application
- having separate analysis programs, that 'with' each type
- putting representation clauses (rep specs) on every type
- coding to assumptions made about the physical representation

Each of these methods impose severe overhead or restrictions. With ASIS emerging as an enabling technology, a generalized data analysis capability has been achieved which solves the problem in this domain without the overhead or restrictions.

The concept for ASIS came out of the Software Technology for Adaptable Reliable Systems (STARS) project and facilitates this generalized solution to the off-line data analysis problem. ASIS consists of a set of Ada package specifications, whose underlying implementations, provided by the compiler vendor, allow access to the semantic information contained in the Ada compiler's compilation database. Since the Ada types are compiled into the Ada library, ASIS can be used to query the Ada library and retrieve the information concerning the exact structure of the Ada type. ASIS is currently sponsored by the Ada Joint Program Office (AJPO) to become an industry standard for accessing semantic information from Ada compiler libraries. Having ASIS as a standard will facilitate the development of portable ASIS applications.

The generalized data analysis capability is achieved by reading the data from secondary storage, using ASIS to determine the structure of the Ada type associated with the object, and converting the data into strongly typed primitives or displayable images.

BACKGROUND

This technique for data analysis has been successfully implemented by IBM for the System 1 and Brilliant Eyes contracts. These contracts deal with ground station processing of satellite data. IBM's Advanced Automation System (AAS) contract for the FAA has also implemented similar ASIS applications.

Presented at the Software Technology Conference (STC'93) on 20 April 1993 in the Software Technology Track, Track #4.
Please see companion Slide Presentation available through the ASIS Home Page at the **http://www.acm.org/sigada/WG/asiswg/asiswg.html** URL.
Attachments 1,2, & 3 are provided with the Slide Presentation as slides 15, 16, & 17.

The real time component of the satellite ground station consists of software modules which are virtual nodes. These nodes process the satellite data in a distributed environment communicating with each other via messages <attachment 1>. A message object is an instance of some (usually complex) Ada type. As messages are passed between nodes, they can also be written to a log file on tape or disk.

A function of the non-real time component of the ground station is delogging and analyzing the data from the log file. This function is required to be a universal delogger; one program capable of delogging any message object from the real time system. The final solution for implementing this universal delogger began to unfold with the introduction of ASIS in early 1992.

The STARS project had been developing the ASIS concept since 1990. Accessing the semantic information stored by the Ada compiler in the Ada library is useful for tool development in the Ada environment. These tools include such things as metrics analyzers, standards checkers, reverse engineering tools, and document generation tools. Since the implementation of the Ada library is compiler dependent, all tools requiring the Ada library semantic information have been tied into those proprietary interfaces. The goal of ASIS was to abstract the interface to the Ada library such that the interface was independent of the compiler vendors' Ada library implementation. This would increase productivity for tools development and open the door for tools to be portable across environments of compiler vendors who implement ASIS. ASIS was prototyped in 1991, and released as a product by Telesoft in January 1992.

In looking at ways to meet the requirement for a universal delogger, IBM evaluated the ASIS interface in April 1992. Some additional functionality was needed to provide representation attribute information and to handle dynamic semantic information for discriminated record types. The extended functionality was provided by Telesoft, and IBM began converting the delogger application in May 1992. The production version was completed in September 1992.

TECHNICAL APPROACH

The log file contains a chronological sequence of logged messages of various object kinds <attachment 2>. A header is attached to each logged message that identifies the object kind for the application data following. The object kind corresponds to an Ada type, usually a record type. The record type may take full advantage of the Ada syntax. It may be private, discriminated, contain nested arrays, nested records, instantiations of generics, other private types, or any combination of the above <attachment 3>.

The delogger reads the header to obtain the object kind, then recurses through the type structure calling ASIS functions to unravel the structure and obtain position information for the type components. In this manner the delogger determines the primitive type (integer, float, string, enumerated) and position of each element in the logged message.

The application data has been read in from secondary storage as an array of bytes. The delogger can now use the position information to index into the byte array, and the primitive type information to convert those bytes into a strongly typed primitive object. This primitive object can then be used in analysis algorithms, displayed, or printed to a report. The following is a simplified example to illustrate the dialog that occurs with ASIS. Many details have been omitted. The purpose is to provide exposure to the kind of processing involved in using ASIS.

Assume that the real time system declares an object of TEST_TYPE1 <attachment 3>:

- --COUNT contains some integer value

- TEST_OBJECT: TEST_PACKAGE.TEST_TYPE1 (DISCRIM1= >COUNT);

TEST_OBJECT is then populated with data. An unchecked_conversion is performed to convert TEST_OBJECT to a byte string. A header is appended to the byte string. A field in the header indicates that the application data in the byte string is of type TEST_PACKAGE.TEST_TYPE1 <attachment 2>. The header and application data byte string are then written to the log file as a log record. This all occurs within the real time system.

In a non-real time mode, the universal delogger is executed. It reads the log record from the log file, and separates the header from the application byte string. Using header information, the delogger discerns that the application data is of type TEST_PACKAGE.TEST_TYPE1. The delogger then initiates a dialog with ASIS to determine the data structure for TEST_TYPE1.

The first queries involve getting from ASIS a list of declared items in the package spec.

- COMPILATION_UNIT : = ASIS_COMPILATON_UNITS.LIBRARY_UNIT (NAME = > 'TEST_PACKAGE',...

- THE_PACKAGE_DECLARATION : = ASIS_COMPILATION_UNITS.UNIT_DECLARATION (A_COMPILATION_UNIT = > COMPILATION_UNIT);

- DECLARED_ITEMS : = ASIS_DECLARATIONS.VISIBLE_PART_DECLARATIVE_ITEMS (PACKAGE_SPECIFICATION = > THE_PACKAGE_DECLARATION);

The list of DECLARED_ITEMS is searched to find the declaration for TEST_TYPE1. We now have TYPE_DECLARATION:

- type TEST_TYPE1 (DISCRIM1: INTEGER) is private;

Once the declaration is in hand, the 'kind' of declaration needs to be determined. From the Ada Language Reference Manual (LRM) appendix E:

- type_declaration :: = full_type_declaration | incomplete_type_declaration | private_type_declaration

ASIS is queried to determine the declaration kind:

- DECLARATION_KIND := ASIS_DECLARATIONS.KIND (TYPE_DECLARATION);

So DECLARATION_KIND is:

- private_type_declaration

If DECLARATION_KIND is an incomplete_type_declaration, or a private_type_declaration (which it is), then we need to get the FULL_DECLARATION:

- FULL_DECLARATION := ASIS_DECLARATIONS.CORRESPONDING_TYPE_DECLARATION (TYPE_DECLARATION);

FULL_DECLARATION is:

- type TEST_TYPE1 (DISCRIM1 : INTEGER) is

```
record
        FIELD1 : RANGE_TYPE;

        FIELD2 : ARRAY_TYPE
            (1..DISCRIM1);

end record;
```

Analysis now proceeds on the full_type_declaration. From the LRM appendix E:

- full_type_declaration ::= TYPE identifier [discriminant_part] IS type definition;

We now query ASIS for the type definition:

- TYPE_DEFINITION := ASIS_DECLARATIONS.TYPE_DECLARATION_DEFINITION (FULL_DECLARATION);

TYPE_DEFINITION is:

- record

  FIELD1 : RANGE_TYPE;

  FIELD2 : ARRAY_TYPE (1..DISCRIM1);

  end record;

We then query ASIS for the 'kind' of type definition:

- TYPE_KIND :=ASIS_TYPE_DEFINITIONS.KIND (TYPE_DEFINITION);

TYPE_KIND is:

- a_record_type_definition.

We then get from ASIS the list of declared record components:

- COMPONENT_LIST := ASIS_TYPE_DEFINITIONS.RECORD_COMPONENTS (TYPE_DEFINITION);

We then analyze each one in the list. The first COMPONENT is:

- FIELD1 : RANGE_TYPE;

We need to evaluate the type of this field, so we query ASIS for the subtype indication, which is:

- RANGE_TYPE.

We then get from ASIS the declaration for RANGE_TYPE, which is:

- subtype RANGE_TYPE is POSITIVE range 10..20;

We query ASIS for the 'kind' of declaration again, and ASIS returns:

- a_subtype_declaration

We get from ASIS the type definition

- POSITIVE range 10..20;

ASIS is now queried for the base type (see LRM 3.3(4)):

- BASE_TYPE := ASIS_TYPE_DEFINITIONS.BASE_TYPE (TYPE_DEFINITION);

BASE_TYPE is:

- INTEGER

INTEGER is a primitive type. So we now know that FIELD1 is an integer. We also get from ASIS the position of FIELD1 (see LRM 13.7.2 Representation Attributes):

- POSITION
- FIRST_BIT
- LAST_BIT

With this information, we can now index into the application byte string, extract the appropriate bits for FIELD1, convert the bits into an integer, and display the integer value to the user.

LESSONS LEARNED

Functionally, the universal delogger using ASIS is completely successful. It eliminated 100K SLOC that was generated using the previous approach, and saved 88 labor days per system build over the previous approach.

The peripheral issues involve configuration management, versioning, and performance.

Since the Ada library serves as a database for ASIS, the library must persist and be accessible at run time for the universal delogger. Traditionally, the Ada library is only used for compiling and binding. It now becomes a system configuration item released as an integral part of the functioning non-real time component of the system.

In addition, the version of software used in the real time system to create the log file must match the version of the Ada library accessed by the delogger at run time. This means that versions of log files and versions of Ada libraries must be tracked and correlated. Also, the Ada library may be large and supporting multiple versions on line may be impractical.

Performance is reasonable considering the nature of the application. If accessing the log file or Ada library across a network, excessive network traffic can be generated and become a bottleneck. Significant performance improvement can be realized by caching the retrieved semantic information in memory. This avoids ASIS queries for subsequent log records of the same object kind, but may require large amounts of memory depending on the characteristics of the data in the log file. Although techniques have been explored to increase performance, it can never be expected to equal a delogger application that 'withs' the type and does an unchecked_conversion of the byte string into the type.

CONCLUSION

The use of ASIS in implementing a universal delogger is a prime example of leading edge technology emerging as a concept from STARS and becoming a reality in current systems. As ASIS becomes an industry standard, future projects will benefit even more from this pioneering technology.