"Transitioning an ASIS Application: Version 1 to Ada95 2.0"

Joseph R. Wisniewski
Commercial Software Solutions, Ltd.
13513 Esworthy Rd. Suite #101
Darnestown, Md 20874
877-538-0136
wisniew@acm.org

ABSTRACT

ASIS (Ada Semantic Interface Specification) applications written to the "Version 1" standard or subsequent 1.X permutations of this standard (such as Version 1.1.1) were developed with and for Ada83. Porting such applications to the Ada95 ASIS standard, Version 2.0, is a non-trivial task. Such an effort needs to address the major changes to the semantic abstractions of the language as represented by the Version 2.0 Ada95 ASIS specification.

This paper is an experience report that documents an approach used to port several Version 1.0 ASIS applications to Version 2.0 compliance. A methodology for "finding" the mappings between the two specifications is documented along with the results (a mapping table of the ASIS queries) of that approach.

Keywords

ASIS, Element, Traverse_Element, Semantics, Syntax, Context, ASISWG

1. INTRODUCTION

Porting applications, in general, is almost always more difficult than planned, regardless of language. Unintentional, or intentional for that matter, host, target, compiler or general environment dependencies almost always have to be addressed at some level. The porting of ASIS applications from the "original" 1.X standard to the Ada95 2.0 standard offers new and additional "porting" challenges.

From a semantic definition standpoint, the Ada95 language did more than just "add on" new features to Ada83. The definitions and interrelationships of the semantics, which are represented by ASIS, underwent architectural changes. Applications that used ASIS for Ada83 therefore, must undergo major changes when being ported to an Ada95 environment.

More simply put, if an Ada vendor replaces Version 1.0 with Version 2.0, the applications that used the 1.0 ASIS interface must now compile against the Version 2.0 ASIS.

The author was involved in porting several of these Ada83 ASIS applications. This paper is a result of that work. Note that this paper is not meant to be an introductory ASIS tutorial. Although there will be discussions of the architecture of the ASIS interface, it is not the purpose of this paper to discuss "first principles" of ASIS. However, this experience report will address the following issues:

- Potential ASIS applications
- History of ASIS
- Specific ASIS applications that were ported that provided a basis for this experience report
- Architectural overview of ASIS Version 1.0
- Architectural overview of ASIS Version 2.0
- Discussion of the differences between the two architectures
- Mechanics of performing the port
- Documentation/Mapping of selected queries

Note: This port was done using Version 1.0 of ASIS by OCSystems for the AIX platform to be ported to the 3.12a2 release of ACT (Ada Core Technologies)/GNAT ASIS for Solaris.

2. POTENTIAL ASIS APPLICATIONS

ASIS is an important "toolset" that can be used for a variety of applications. Some potential applications are:

- Calling tree
- Standards checker
- Types dictionary
- Debugger
- Dependency tree analysis
- Quality Assessment
- Off-line data decoders

All of these applications (and others) depend ultimately on an analysis of the syntax and semantics of the language. Perhaps the best way to understand the benefits of ASIS is to consider the design implications of implementing such applications without ASIS.

In order to implement an application such as a calling tree or a standards checker, in a portable/compiler independent fashion, the developer would by necessity be forced to analyze or parse the source code. (Note: I am ignoring the option of analyzing any compiler generated files including libraries, if there are any, since portability is often a prime requirement for these types of applications.)

Parsing source code in order to implement such applications is formidable, if not impossible, in a practical sense. Several challenges immediately come to mind. In the case of a calling tree, for example, determination of the origin of overloaded subprograms could be extremely difficult. Differentiation between function calls and variable occurrence can be extremely difficult. If use clauses are introduced, the determination of the origin of subprograms gets very complicated.

In short, many of the tasks that such "source code parsers" would have to perform have already been done by the compiler. Does it not make more sense for the compiler to answer these queries, through an interface?

3. HISTORY OF ASIS

ASIS is a published international ISO standard. The name of the standard is ISO/IEC 15291:1999. The Ada special interest group of the ACM (Association of Computing Machinery) chartered a working group, the ASISWG, in 1993 to develop and standardize an interface that would allow access to the semantics and syntax of the language as it applies to some given Ada source code. There have been several revisions to published ASIS standards. The current standard, as it applies to Ada95 is known as ASIS 2.0.

The Version 1.0 specification was derived from what was known as the version 0.4 draft standard, which was developed by TeleSoft.

Subsequent to the Version 1.0 release, and as with most standards, there has been a revision process during the lifetime of the Ada language. There was an initial 1.0 implementation, as noted above, a 1.1.1 standard and the current 2.0 standard that were developed and modified by various Ada vendors. Generally speaking, and in order to avoid confusion, the 1.X standards applied to Ada83 implementations and the current 2.0 and potentially future 2.X implementations apply to Ada95. Despite the fact that the interfaces were "standards", not all Ada vendors implemented all of the standards. This is true of the current 2.0 standard. Not all Ada vendors have implemented the full 2.0 standard, although several vendors are currently at various points in the development cycle for 2.0.

Even within the Ada community, ASIS has not been a very well known concept. That is beginning to change through a variety of networking, marketing and general information exchange forums:

- Comp.lang.ada newsgroup

  This forum is periodically used to field ASIS questions. It is a very good place for a "newbie" to post questions.

- Papers and conference talks/tutorials

  The SIGAda (formerly Tri-Ada) annual conference always has several ASIS tutorials and/or papers, usually presented or with active involvement from key ASIS vendor implementers.

- ACM/SIGAda website

  There is now a dedicated area for ASIS vendors and toolsmiths to advertise their wares.

http://www.acm.org/sigada/wg/asiswg/asiswg.html

- ASIS mailing lists.

  Again, through ACM there are listserver mailing lists of ASIS practitioners of varying levels of interest. This forum has tremendous potential to disseminate ASIS information.

http://www.acm.org/sigada/wg/asiswg/asiswg.html

4. PORTED APPLICATIONS

There are two main applications that provide the basis of this experience report. There were several other very small ASIS applications that also were ported, but provide no additional information and will not be discussed. These two

applications are a standards checker and a types descriptor service.

4.1 Standards Checker

This ASIS application has built-in rules (which can be added to, deleted, and turned on and off) that are responsible for checking for a variety of coding standards. When violations of these rules are encountered in source code, a violation is printed "within" the source code, but only in a report, not in the "original" source code.

Some rules apply to single units, while some rules are applicable to a "family" of units. For example, each unit needed a header, which requires very specific fields in the header. This "header check" is applied to all subprograms and files. There are other much more complex rules that span multiple units and even multiple files. For example, there are rules to detect unused declarations, unnecessary "with" clauses, and misplaced "with" clauses. Continuing with a further description of these rules, in order to determine whether a given declaration is unused (variable/constant declaration, type declaration, input parameter declaration, etc.), it is necessary to interrogate not only the unit that contains the declaration but all subunits. This requires that all such subunits (which could be declared as separates) be contained in the context. Detection of unused or misplaced "with" clauses is equally complex. An unused "with" clause violation occurs when a "with" clause occurs with a given compilation unit but is not used in THAT compilation unit AND in any subunits. A misplaced "with" clause violation occurs when a "with" clause occurs in a given compilation unit and it is not referenced in that compilation unit but IS referenced in a subunit. Again, the relevant subunits need to be in the context in order to completely execute the testing of this rule. In addition, if there are any subunits missing from the context for such a rule, it would be a very good idea for the ASIS tool to detect this condition and report it to the user. Otherwise, the user would understandable be misled when a violation is detected when it could potentially be a "false failure" of such a rule since a missing subunit may have the only references to a potentially missing context clause or declaration.

4.2 Types Descriptor Service

The other major tool that has been ported is a types descriptor service. This ASIS application is responsible for determining the layout of any given set of Ada types. If the layout for a given type can be determined, then this ASIS application generates sizing information for the type. An internal set of rules is initially applied to each candidate type. For example, the layout for some types would not be calculated, such as for nested variant records (variant records that contain variant records), and others. If the overall layout can be determined, then the structure of the type (usually a record at the highest level) is interrogated field by field to determine exactly how the type is "laid out".

One use of such a tool is to be able to "walk the bits" of a data object of such a type. This scenario occurs when "off-line" applications need to decode a "saved" Ada object, represented by a series of bits in a file. This types descriptor application would have saved the "layout information" in a file that such an off-line application could use, along with the "object bits" to ultimately decode those bits. A key element of the ASIS interface, necessary to perform such a task is the "Data Decomposition Annex", as it is known in Version 2.0.

5. OVERVIEW – VERSION 1.0

The ASIS Version 1.0 interface (again, as implemented in OCSystems compiler for IBM AIX) is comprised of the following packages:

ASIS_Compilation_Units

ASIS_Declarations

ASIS_Elements

ASIS_Ada_Environments

ASIS_Libraries

ASIS_Names_and_Expressions

ASIS_Representation_Clauses

ASIS_Statements

ASIS_Text

ASIS_Type_Definitions

ASIS_Ada_Program

In the interest of limiting duplicate discussions and concepts, only the highest-level package architecture of Version 1.0 will be discussed in this section. The details/constructs of those packages that "carry over" into the 2.0 architecture will be discussed in the architecture comparison section. The Version 2.0 architecture will go into much greater detail as to the contents of the packages and how these relate to and influence the overall ASIS architecture, since Version 2.0 is the "current" release of ASIS, the internal structures of which are more important to understand that Version 1.0.

The 1.0 architecture is rather flat. There are some nested packages in the architecture that are not shown, because they are not an influence on the main architecture of Version 1.0 ASIS.

The packages as listed represent the main abstractions of ASIS for this version of the interface. Queries that extract semantic state information from given abstractions are

contained in packages that are named as such. For example, packages that extract information from and about expressions are contained in ASIS_Names_and_Expressions. Queries dealing with declarations are contained in ASIS_Declarations, and so forth. Resident in most of the packages is a "kind" enumeration type that identifies the different "kinds" of each abstraction. For example, contained within the package ASIS_Declarations is the following abbreviated type:

```
type Declaration_Kind is
        ( A_Variable_Declaration,
          A_Component_Declaration,
          A_Constant_Declaration,
          A_Deferred_Constant_Declaration,
          A_Generic_Formal_Object_Declaration,
          A_Discriminant_Specification,
          A_Parameter_Specification,
          …);
```

This type is used both by the ASIS application and indirectly by the ASIS implementation.

6. OVERVIEW – VERSION 2.0

ASIS Version 2.0 is comprised of a series of packages with package ASIS being the root package and the remaining packages comprising a package/child package hierarchy:

```
Asis
        Asis.Ada_Environments
                Asis.Ada_Environments.Container
        Asis.Clauses
        Asis.Compilation-Units
                Asis.Compilation-Units.Relations
                Asis.Compilation-Units.Times
        Asis.Declarations
        Asis.Definitions
        Asis.Elements
        Asis.Errors
        Asis.Exceptions
        Asis.Expressions
        Asis.Extensions
        Asis.Ids
        Asis.Implementation
                Asis.Implementation.Permissions
        Asis.Iterator
        Asis.Statements
        Asis.Text
        Asis.Data_Decomposition
```

6.1 ASIS

Package ASIS supplies types that define the abstractions represented within ASIS.

As noted, package ASIS is the "root" package of the ASIS package hierarchy. Package ASIS supplies visible types that provide definition and distinction to the different abstractions represented by the entire ASIS interface.

The following types are provided by package ASIS. They are used by any and all child packages of package ASIS and any ASIS applications. A thorough understanding of package ASIS as it relates to the abstractions represented by the types supplied is critical to an overall understanding of ASIS and subsequent abilities to write ASIS applications.

- type ASIS_Integer

  ASIS_Integer is the base "integer" abstraction. It is a subtype of an implementation defined integer type that is also declared in package ASIS.

```
subtype Implementation_Defined_Integer_Type is integer;
```

```
subtype ASIS_Integer is Implementation_Defined_Integer_Type;
```

- type ASIS_Natural

  ASIS_Natural is the expected constrained subtype of ASIS_Integer

```
subtype ASIS_Natural is ASIS_Integer range 0..ASIS_Integer'last;
```

- type ASIS_Positive

  ASIS_Positive is the expected constrained subtype of ASIS_Integer

```
subtype ASIS_Positive is ASIS_Integer range 1..ASIS_Integer'last;
```

- type Element

- type List_Index
- type Element_List

Type Element is an undiscriminated private type. It is an abstraction representative of an Ada lexical element.

```
type Element is private;
```

ASIS has supplied an abstraction for a list, specifically an element list. A single element or a list of elements can be returned by an ASIS query. In addition, ASIS applications may need to create lists of elements returned by various ASIS queries

List_Index is a subtype of ASIS_Positive

```
subtype List_Index is ASIS_Positive range
        1..
Implementation_Defined_Integer_Constant
```

where Implementation_Defined_Integer_

Constant is just that; a constant defined by the implementation, here in package ASIS.

An element_list is an array of elements indexed by list_index

```
type Element_List is array
   (List_Index range <>) of Element;
```

- Element subtypes/Element List subtypes

This is another crucial concept, necessary for the understanding of ASIS. Each abstraction represented in ASIS has a subtype declared in package ASIS. From a "compilation/strong typing" point of view, these subtypes do not imply a strong typing model. That is, an ASIS query that is expecting a "declaration", must be passed in an element that represents a declaration. However, that parameter can technically be declared as any kind of element subtype, although this certainly would not help the understanding of such an application!

Likewise, for most all element subtypes there are corresponding element list subtypes (those will not be listed here, due to this redundancy).

The element subtype declarations are:

```
subtype Access_Definition is Element;
subtype Association is Element;
subtype Case_Statement_Alternative is
      Element;
subtype Clause is Element;
subtype Component_Clause is Element;
subtype Component_Declaration is
      Element;
subtype Component_Definition is
     Element;
subtype Constraint is Element;
subtype Context_Clause is Element;
subtype Declaration is Element;
subtype Definition is Element;
subtype Discrete_Range is Element;
subtype Discrete_Subtype_Definition is
     Element;
subtype Discriminant_Association is
    Element;
subtype Defining_Name is Element;
subtype Exception_Handler is Element;
subtype Expression is Element;
subtype Formal_Type_Definition is
    Element;
subtype Generic_Formal_Parameter is
    Element;
subtype Identifier is Element;
subtype Name is Element;
subtype Parameter_Specification is
   Element;
```

```
subtype Path is Element;
subtype Pragma_Element  is Element;
subtype Range_Constraint is Element;
subtype Record_Component is Element;
subtype Record_Definition is Element;
subtype Representation_Clause is
   Element;
subtype Root_Type_Definition is
   Element;
subtype Select_Alternative is Element;
subtype Statement is Element;
subtype Subtype_Indication is Element;
subtype Subtype_Mark is Element;
subtype Type_Definition is Element;
subtype Variant is Element;
```

Close inspection of these subtype declarations reveals the abstractions that form the architectural basis of ASIS.

- type Compilation_Unit/Compilation_Unit_List

  The type compilation unit is a private type, as is element. A compilation consists of a context clause, which may contain any number of (including none) "with" clauses, "use" clauses and "use type" clauses and then a declaration of a library_unit or item and finally pragmas that apply to the compilation unit.

```
type Compilation_Unit is private;
```

```
type Compilation_Unit_List is array
 (List_Index range <>) of
Compilation_Unit;
```

- Unit Kinds (set of enumeration types)

  Many of the abstractions listed above in the element subtypes have "kinds" associated with them. Likewise, compilation_units are characterized by their "kind" (abbreviated to unit_kinds)

- Element and element subtype kinds

  The element subtypes define the different ASIS abstractions. These abstractions however, need to be further categorized and differentiated. This is done via a set of enumeration types that define many of the aforementioned abstractions. For example: the major kinds that exist describe what can be considered as the "high level" abstractions:

  Elements → Element_Kinds
  Declarations → Declaration_Kinds
  Definitions → Definition_Kinds
  Type_Definitions → Type_Kinds
  Statements → Statement_Kinds
  Expressions → Expression_Kinds

  There are many more enumerations/kind types in package ASIS, however these 6 abstractions/kinds are used most often in a typical ASIS application.

- Type Traverse_Control

```
type Traverse_Control is
   (Continue,
    Abandon_Children,
    Abandon_Siblings,
    Terminate_Immediately);
```

  This enumeration type provides traversal control for a generic procedure that is provided in the ASIS.Iterator package specification.

- Subtype Program_Text

  (subtype of Wide_String)

  This type simply provides ASIS with its own string abstraction.

6.2 ASIS.ADA_ENVIRONMENTS/ ASIS.IMPLEMENTATION

Queries in ASIS.Ada_Environments and ASIS.Implementation occur at initialization of an ASIS application. ASIS needs to know what Ada code it is going to operate on. These identified compilation units are gathered together in what is called a "context'. A context can be created and then deleted and another context created within a given ASIS application. The following queries are a sampling of those that are generally used when initializing an ASIS application:

(From ASIS.Implementation)

```
procedure Initialize;
```

(From Asis.Ada_Environments)

```
procedure Associate
   (The_Context : in out Asis.Context;
    Name : in Wide_String;
    Parameters : in Wide_String := Defalt_Parameters);
```

This procedure associates a name with a context. The parameters are interpreted in a compiler dependent fashion to know what files to include in the context.

```
procedure Open
  (The_Context : in out Asis.Context);
```

This procedure is called after the associate call and actually "populates" the context and makes it ready for use.

```
procedure Close
  (The_Context : in out Asis.Context);
```

This procedure closes the ASIS context.

6.3 ASIS.ITERATOR

This is a critically important package in the ASIS architecture. This package supplies a generic that is used to traverse elements.

```
generic
   type State_Information is limited private;

   with procedure Pre_Operation
      (Element : in Asis.Element;
       Control : in out Traverse_Control;
       State : in out State_Information) is <>;

   with procedure Post_Operation
      (Element : in Asis.Element;
       Control : in out Traverse_Control;
       State : in out State_Information) is <>;

procedure Traverse_Element
      (Element : in Asis.Element;
       Control : in out Traverse_Control;
       State: in out State_Information);
```

This procedure accepts an element (viewed as a starting point element) and then recursively deconstructs the element into its constituent element tree and traverses that tree. Pre-Operation is a user-supplied routine as is Post_Operation. As their names indicate, these routines are responsible for performing the actual "work" during the traversal immediately prior to and immediately after encountering some traversed sub-element.

Processing control is maintained by the Control parameter. This is set in the pre and post operation routines to determine traversal control. The possible values of traverse_control are Continue, Abandon_Children, Abandon_Siblings, Terminate_Immediately. How this parameter is set, depends upon exactly what the application is trying to do and what is detected in the pre and post operation routines.

State_Information is a user-supplied type that maintains application-necessary information outside of the scope of the instantiation of the generic. This is a very useful aspect of this generic.

6.4 ASIS.ELEMENTS

Package ASIS.Elements supplies general purpose element processing queries. It also supplies a "kind" query that returns the "kind" of a given abstraction.

For example:

```
function Element_Kind (Element: in Asis.Element)
         Returns Asis.Element_Kinds;

function  Declaration_Kind
      (Declaration : in Asis.Declaration)
         Return Asis.Declaration_Kinds;
```

Package ASIS.Elements also supplies special "Compilation Unit to Element" gateway functions.

For example:

```
function Unit_Declaration
  (Compilation_Unit: in Asis.Compilation_Unit)
    return Asis.Declaration
```

This function returns the declaration of a given compilation unit.

```
function Enclosing_Compilation_Unit
  (Element : in Asis.Element)
    return ASIS.Compilation_Unit
```

This very important function returns the compilation unit where a given element "exists".

```
function Context_Clause_Elements
  (Compilation_Unit : in Asis.Compilation_Unit;
   Include_Pragmas : in  boolean := false)
     Return ASIS.Context_Clause_List;
```

This function returns all with clauses, use clauses and use type clauses associated with the compilation unit.

6.5 ASIS.COMPILATION_UNITS

This package contains queries that define the "compilation unit" abstraction. Some of these queries are:

```
function Unit_Kind
  (Compilation_Unit : in Asis.Compilation_Unit)
    return ASIS.Unit_Kinds;
```

This function determines exactly what kind of compilation unit this is.

```
function Unit_Class
  (Compilation_Unit : in Asis.Compilation_Unit)
    return ASIS.Unit_Classes;
```

This function returns a class enumeration associated with the compilation unit. "Class" represents a classification of compilation units that address characteristics of public, private, body and subunit.

```
function Compilation_Unit_Body
  (Name : in Wide_String;
   The_Context : in Asis.Context)
     Return Asis.Compilation_Unit;
```

This function returns the library_unit_body or subunit with the name.

```
function Compilation_Units
  (The_Context : in Asis.Context)
    return Asis.Compilation_Unit_List;
```

This function is often a starting point in an ASIS application. After a context is created, this query returns all of the compilation units associated with the context. One can now loop through the compilation units and "process" as the application dictates.

6.6 ASIS.EXTENSIONS

Package ASIS.Extensions is a vendor supplied package that includes "higher level" queries; common ASIS routines if you will. These queries are not part of the "standard", for whatever reason, but have been deemed "common or necessary enough" to be supplied by the vendor instead of written by the application developers as common ASIS routines.

6.7 ASIS.DATA_DECOMPOSITION

This package is generally responsible for the difficult task of determining object sizes and other bit-order related tasks, as mentioned previously. It is not a "required" package in the Version 2.0 interface.

6.8 ABSTRACTION PACKAGES

Most of the rest of the child packages of package ASIS represent abstract data types that represent the abstractions for which the package is named. These include:

- ASIS.Expressions
- ASIS.Declarations
- ASIS.Clauses
- ASIS.Text
- ASIS.Statements

These packages, in general, contain two types of queries, those that maneuver within the abstraction and those that are gateway queries to other abstractions.

7. COMPARISON 1.0 → 2.0

The similarities between the interfaces are:

- Separate packages are identified to represent different semantic abstractions
- Enumeration types exist for each of the semantic abstractions to differentiate between "flavors" of the given abstraction and to provide a method to determine the appropriate abstraction query to invoke.
- The interface itself is a "broad" interface; that is, the responsibility is placed on the application to "know" the appropriate query or sets of queries to invoke. This is done partly through the use of the "kind" query that exists for each of the major abstractions.
- Both ASIS versions utilize the main concepts of an "element" and "element subtyping".

The main differences between the two versions are:

- The concept of a library has been replaced with that of a context
- Ada95 syntax elements have been added
- Abstractions have been added and changed. Some of these are:
  - Names_and_Expressions are now expressions
  - Representation_Clauses has been incorporated into a larger abstraction of "clauses"
  - The concept of "Choices" with an accompanying "Choice_Kind" has been changed
- The previous concept of "type definition" has been replaced and expanded/changed into two separate abstractions of "definitions" and "types" along with other subservient abstractions such as "traits".
- The "root" ASIS package now contains all of the "kind" enumerations
- The ASIS.Elements package now contains all of the "element subtypes". Version 1.0 contained a duplicate copy of each subtype in each package that operated on a given subtype. So for example, any ASIS package that may return a "declaration" had a "declaration element" subtype definition.
- Some queries were eliminated in the Version 2.0 interface. Version 1.0 contained boolean queries, such as Is_Constrained. These were mostly eliminated and left to be part of a "higher level of abstraction" to be implemented as distinct from the core implementation.

8. MECHANICS OF THE PORT

Three major activities comprise the porting effort. That is, mapping the packages, mapping the "kind" enumerations and mapping the queries.

8.1 Mapping the Packages

Generally speaking (and accounting for differences mentioned previously):

Version 1.0		**Version 2.0**
ASIS_Compilation_Units	→	ASIS.Compilation_Units (And child packages)
ASIS_Declarations	→	ASIS.Declarations
ASIS_Elements	→	ASIS.Elements
	→	ASIS.Iterator
ASIS_Environment	→	ASIS.Ada_Environments (And child package)
ASIS_Libraries	→	ASIS.Ada_Environments (And child package)
ASIS_Names_and_Expressions	→	ASIS.Expressions
ASIS_Representation_Clauses	→	ASIS.Clauses
ASIS_Statements	→	ASIS.Statements
ASIS_Text	→	ASIS.Text
ASIS_Type_Definitions	→	ASIS.Definitions
ASIS_Ada_Program	→	ASIS
ASIS_Objects_Attribute_Extension		

→ ASIS.Data_Decomposition

This mapping however does not directly account for

- ASIS.Errors
- ASIS.Exceptions
- ASIS.Ids
- ASIS.Implementation

These packages were added as a direct by-product of the different abstraction representation in 2.0.

Keep in mind that this package "mapping" is very loose mapping. It is a starting point when attempting to begin mapping the queries and the types that support the queries.

8.2 Mapping the "Kind" Enumerations

Along with mapping the queries, this is where the "rubber meets the road", so to speak. Remember that the "kind" enumeration types, under Version 1.0 resided each in its own abstraction package, along with the queries. Now, under Version 2.0, the "kind" enumeration resides in package ASIS, with the "kinds" query residing in ASIS.Elements. It is beyond the scope of this paper, and the author's experience to be able to map each and every enumeration from each and every "kind" enumeration type. A few comments are in order however.

The initial difference between the 2 versions of the "kind" types is due to "Ada95-isms". This is, per se, not a porting issue, but in actuality, since the port "is" to an Ada95 compiler, these additional enumerations must be addressed. It is the strong suggestion of the author, that any incorporation of "Ada95-isms" into an ASIS application occur as an activity separate from the actual port to 2.0.

In general, many of the enumerations, taken in their entirety, map one-for-one, with small differences in the actual name of the enumeration. However, there are some cases that are more difficult. One example, for the sake of illustration, is in the case of type definitions and definitions. Under Version 1.0, there is an enumeration "Type_Definition_Kinds". This enumeration type accounted for differentiating between records, arrays, enumerations, floats, integers, private, limited private, etc. Now however, this "type_definition" abstraction has migrated into several layers of abstractions under 2.0. In order to be able to port ASIS application code that previously referenced "type_definition_kind", one now has to deal with

- Definition_Kind
- Type_Kind
- Trait_Kind

And potentially other "kind" types.

8.3 Mapping the Queries

There is no magic to determining the query mapping. The process that was followed was basically to first look in the package mapping from above, for a query identical to the 1.0 version of the query. If one did not exist, then an attempt to look for one similar in name occurs. If that fails, then an attempt to find queries that accepted the same "Appropriate element and sub-element kind" as documented in the commentary of the queries would proceed. It is this action that yielded the most success. Many queries had changes to the name of the queries but did basically the same function.

If this attempt failed, as it did in some cases, then a query-by-query search of the entire interface would proceed. If that failed also, then it is likely that there is no single "query mapping" available. In this case, a utility has to be written to combine existing queries. This activity can also be non-trivial. Some examples of this kind of a "more challenging" utility are the ones for ground_type and parent_type that one would have to write to emulate Version 1.0 functionality.

This effort of "query mapping" is by far the most difficult aspect of a Version 1.0 → Version 2.0 port. This is why the following partial mapping table has been included in this paper. It is the documentation of the mapping effort. This mapping is not complete and no warranties are implied. At the very least, it will provide a major jump-start to any 1.0 → 2.0 port.

9. QUERY MAPPINGS (SELECTED)

(Legend)

*	→ Abbreviated
"Same"	→ Same query in titled package
(Simple package name)	→ Same name, new package
"No mapping"	→ No mapping found or none supplied by vendor
"Utility"	→ No direct mapping
"None Necessary"	→ Query will never be needed
package.query	→ New package/new query

ASIS_Compilaiton_Units	ASIS.Compilation_Units
Can_Be_Main_Program	Same
Comp_Command_Line_Options *	Same
Compilation_Pragmas	(Elements)
Compilation_Units	Same

Context_Clause_Elements	(Elements)
Corresponding_Library_Unit	Corresponding_Declaration
Corresponding_Secondary_Unit	Corresponding_Body
Enclosing_Compilation_Unit	(Elements)
Enclosing_Library	Enclosing_Context
Is_Consistent	No Mapping
Is_Equal	Same
Is_Nil	Same
Is_Obsolete	No Mapping
Kind	Unit_Kind
Library_Unit	Library_Unit_Declaration
Library_Units	Library_Unit_Declarations
Name	Unit_Full_Name
Referenced_Units	Utility
Secondary_Unit	Compilation_Unit_Body
Secondary_Units	Compilation_Unit_Bodies
Subunits	Same
Subunit_Ancestor	Utility
Time_Of_Last_Update	No Mapping
Text_File_Name	Text_Name
Unique_Name	Same
Unit_Declaration	(Elements)
ASIS_Type_Definitions	**ASIS.Definitions**
Access_To	Access_To_Object_Definition
Base_Type	Declarations. Corresponding_First_Subtype
Choice_Kind	Elements.Definition_Kind/ Elements.Expression_Kind
Choice_Name	No mapping necessary
Component_Subtype_Indication	Same
Constraint_Kind	(Elements)
Discrete_Range_Kind	(Elements)
Discrete_Ranges	Same
Discriminant_Associations	Same
Discriminant_Expression	(Expressions)
Discriminants	Same
Enumeration_Literal_Declarations	Same
Enumeration_Literal_Identifiers	Utility
Ground_Type	Utility
Index_Constraint	Utility
Integer_Constraint	Same
Is_Constrained_Array	Utility
Is_Discriminated	Utility
Is_Predefined	Utility
Kind	Elements.Type_Kind

	Elements.Definition_Kind
Lower_Bound	Same
Parent_Subtype	Parent_Subtype_Indication
Parent_Type	Utility
Real_Type_Constraint	Real_Range_Constraint
Record_Components	Same
Subtype_Constraint	Same
Subtype_Definition_ Subtype_Indication	None Necessary
Type_Declaration_Definition	Declarations. Type_Declaration_View
Type_Definition_Declaration	Elements.Enclosing_Element
Type_Mark	Subtype_Mark
Upper_Bound	Same
Variant_Choices	Same
Variant_Components	Record_Components
Variants	Same
ASIS_Text	**ASIS.Text**
Comment_Image	Same
Element_Span	Same
First_Line_Number	Same
Image	Line_Image/Element_Image
Is_Text_Available	Same
Lines	Same
Non_Comment_Image	Same
Last_Line_Number	Same
ASIS_Statements	**ASIS.Statements**
Accept_Body_Statements	Same
Actual_Parameter	Expressions.Actual_Parameter
Arm_Statements	Sequence_Of_Statements
Block_Body_Statements	Block_Statements
Block_Exception_Handlers	Same
Case_Statement_Alternatives	Statement_Paths
Case_Statement_Alternative_ Statements	Sequence_of_Statements
Declarative_Items	Block_Declarative_Items
Exception_Choices	Same
Goto_Label	Utility
Handler_Statements	Same
If_Statement_Arm_Kind	Path_Kind
If_Statement_Arms	Statement_Paths
Is_Implicitly_Declared	Elements.Is_Part_of_Implicit
Is_Labeled	Utility
Is_Others_Handler	Utility

Is_When_Others	Utility
Kind	Elements.Statement_Kind
Label_Name	Same
Loop_Kind	Elements.Statement_Kind
Loop_Statements	Same
Raised_Exception	Same
Select_Alternative_Statements	Sequence_of_Statements
Select_Statement_Arms	Statement_Paths
ASIS_Representation_Clauses	**ASIS.Clauses**
Associated_Length_Clause_ Representation	Utility
Length_Clause_Attribute_Kind	Utility
Length_Clause_Simple_Expr *	Utility
Kind	Elements. Representation_Clause_Kind
ASIS_Names_And_Expressions	**ASIS.Expressions**
Attribute_Designator_Argument	Attribute_Designator_Expressions
Attribute_Designator_Name	Same
Called_Function	Corresponding_Called_Function
Component_Choices	Array_Component_Choices Record_Component_Choices
Component_Expression	Same
Components	Array_Component_Associations Record_Component_Associations
Converted_or_Qualified_Expr *	Same
Definition	Corresponding_Name_Definition
Expression_Parenthesized	Same
Enclosing_Declaration	Utility
Expression_Type	Utility
Function_Call_Parameters	Same
Id_Kind	Elements.Defining_Name_Kind
Index_Expressions	Same
Is_Implicitly_Declared	Elements.Is_Part_of_Implicit
Is_Predefined	Utility
Is_Referenced	Same
Is_Static	Extensions.Is_Static
Kind	Elements.Expression_Kind
Named_Declaration	Name_Declaration
Named_Packages	Utility
Position_Number_Image	Name_Image
Prefix	Same
References	Same
Selection_Kind	Elements.Expression_Kind
Selector	Same

Static_Value	Value_Image
String_Name	Declarations.Defining_Name_Imag Name_Image Text.Element_Image
Type_Mark	Converted_Or_Qualified_Subtype _Mark
ASIS_Libraries	**ASIS.Ada_Environments**
Associate	Same
Close	Same
Disassociate	Same
Open	Same
ASIS_Environment	**ASIS.Implementation**
Finalize	Same
Initialize	Same
ASIS_Elements	**ASIS.Elements**
Argument_Associations	Pragma_Argument_Associations
Enclosing_Major_Element	Enclosing_Element
Is_Equal	Same
Is_Nil	Same
Major_Element_Kind	Element_Kind
Name	Pragma_Name_Image
Parse_Major_Element	Traverse_Element
Pragmas	Same
ASIS_Declarations	**ASIS.Declarations**
Body_Stub	Correspondig_Body_Stub
Corresponding_Constant_Decl *	Same
Corresponding_Type_Declaration	Same
Discriminants	Definitions.Discriminants
Identifiers	Names
Initial_Value	Initialization_Expression
Is_Body_Stub	Utility
Is_Function	Utility
Is_In_Private_Part	Utility
Is_Initialized	Utility
Is_Operator_Definition	Utility
Is_Package	Utility
Is_Procedure	Utility
Is_Renaming_Declaration	Utility
Is_Spec	Utility
Is_Subprogram	Is_Subunit
Is_Subunit	Same
Is_Task	Utility

Is_Type_Declaration	Utility
Is_Visible	Utility
Generic_Formal_Subprogram_ Default_Kind	Elements.Default_Kind
Generic_Unit_Name	Same
Kind	Elements.Declaration_Kind
Object_Declaration_Definition	Object_Declaration_View
Package_Body_Block	Body_Block_Statement
Parameters	Parameter_Profile
Private_Part_Declarative_Items	Same
Renamed_Base_Entity	Corresponding_Base_Entity
Renamed_Entity	Same
Return_Type	Result_Profile
Type_Declaration_Definition	Type_Declaration_View
Type_Mark	Declaration_Subtype_Mark
Unit_Body	Corresponding_Body
Unit_Specification	Corresponding_Declaration
Visible_Part_Declarative_Items	Same

10. CONCLUSIONS

Porting an ASIS application from Version 1.0 to Version 2.0 is a non-trivial task. The guidance and documentation in this experience report should provide invaluable benefit to such a port. ASIS is a non-trivial solution to non-trivial problems. The use and understanding of ASIS, as this practitioner has found out, is a wonderful "training tool" for a thorough understanding not only of the semantics of the Ada language but the underlying semantics.

11. ACKNOWLEDGMENTS

I would like to express my appreciation of Mr. Tom Fleck of OCSystems and especially to Dr. Sergey Rybin of ACT/GNAT for their support and input while performing the activities that provided the basis for this paper. Both exemplify the professionalism of many that have made it pleasurable to work in the Ada field.

12. REFERENCES

[1] Wisniewski, J.R. Introduction to ASIS (Colorado Springs, CO, July 1999), ASEET99 Conference

13. TRADEMARKS