Reliable interface definition and checking: ANDF for COSE

Stavros Macrakis

Open Software Foundation
Research Institute

May 1993

COSE defines a standard Applications Programming Interface. ANDF technology can specify this interface precisely and check applications' conformance to it. This implementation-independent specification enhances portability.

1. Introduction

COSE is a common application programming interface. Unlike existing *de facto* and *de jure* standards (Posix, XPG/3), it is intended to cover the full interface of commercially available Unix-derived systems.

Current plans call for COSE to be documented in English and defined using parameterized C header files. Parameterized headers cannot be validated for consistency across platforms, nor can applications be validated for portability across conforming platforms.

This paper proposes using OSF's ANDF C to define COSE. ANDF tools can then check that the COSE interfaces are indeed consistent across platforms, and can check that applications are portable across platforms.

2. COSE interfaces in C are parameterized by implementation

COSE interfaces are specified as C definitions

COSE interfaces are currently documented in traditional Unix-style `man` pages. Functional interfaces are described with C prototypes, which may use C built-in types (`int`, `char`, …) or interface-specific types. In the program itself, the prototypes and interface-specific types are defined by `include`'ing a header file (`.h`) specified in the `man` page.

Header files define interface-specific types in terms of built-in types and type constructors, and functional interfaces in terms of the interface-specific types.

COSE's C header files are parameterized

All implementations will probably use the same prototypes to define functional interfaces[1]. However, the interface-specific types will vary. Thus there cannot be a standard set of COSE header files across all implementations. Instead of having completely disjoint header files, however, COSE proposes parameterized header files with appropriate compile-time conditionals (`#ifdef`s). In effect, each platform will have a somewhat different header file, parameterized by its characteristics.

The advantage of this approach is that these parameterized header files can be used with current compilers.

However, they have several severe disadvantages.

Parameterized header files cannot be validated for internal consistency

Since the header files will be complicated, errors will be hard to find, and maintaining them will be difficult.

1. Actually, some functional interfaces may be defined as macros on some implementations. Since macros and functions belong to different namespaces, this complicates matters.

Since the header files are parameterized by platform, tools cannot verify that they are consistent between platforms.

Parameterized header files specify too much

Since the header files can only use the C type system, certain aspects of type behavior have to be made explicit even the published specification leaves them undefined. Applications may take advantage of them (with no checking possible) and be incompatible with some future implementation. The best possible check is against all possible parameterizations of the header file, which may not reflect all the implementation freedom defined by the specification.

Parameterized header files do not specify enough

Since the header files must define prototypes in terms of C types, they cannot specify what the desired semantics are, only one implementation of them. They do not reflect the full intent of the COSE specification.

Parameterized header files do not support conformance checking

The only way to assure that an application is portable to a given COSE platform is to compile it on that platform. An application would have to be compiled on every existing platform for conformance checking, and that would only check currently available platforms.

Parameterized header files restrict future development

Since parameterized header files specify too much about existing implementations, not enough about the interface itself, and do not support conformance checking against the standard, they restrict future implementations to be like present implementations. To validate an application against a future implementation of COSE would require compiling it on that implementation.

3. COSE interfaces in ANDF are constant across implementations

Unlike C header files, ANDF header files specify the interface itself, and not one implementation of it.

COSE interfaces can be specified as ANDF C definitions

Using ANDF, functional interfaces would continue to be specified by prototypes (possibly in token syntax). Types, however, would be abstracted using ANDF mechanisms. Only the properties of a type specified by the COSE standard would be reflected in the header.

COSE's ANDF C header files are implementation-independent

All implementations would use identical header files, with no compile-time ces. Types are specified abstractly; macro and function implementations share the same abstract declaration.

Implementation-specific bindings would be supplied at installation time. The install-time headers would be identical to current platform-dependent header files.

ANDF header files can be validated for internal consistency

ANDF header files are identical across implementations, so the ANDF semantic analyzer can identify any internal inconsistencies or errors.

Since ANDF header files are unparameterized, errors need only be corrected once for all platforms, and extending and maintaining them is easy.

ANDF header files specify exactly enough

ANDF header files specify exactly what the interface requires, no more, no less. Thus applications cannot depend, intentionally or unintentionally, onI

nonstandard features of their development environment. For instance, “internal” structure fields simply do not exist in the ANDF header files.

ANDF header files support conformance checking for implementations

ANDF supports checking of platform-dependent header files against the platform-independent header files. Thus the conformance of target platforms to the COSE standard can be checked independently of particular applications.

ANDF header files support conformance checking for applications

ANDF supports checking of applications against platform-independent header files. Thus the conformance of applications to the COSE standard can be checked independently of particular platforms.

ANDF header files encourage future improvement

Since ANDF header files separate the specification of the interface from its implementation, they are open to innovative implementations. Such innovations are not constrained to be among the alternatives reflected in the parameterized header files.

4. The next steps

COSE should choose between token syntax and tspec syntax

ANDF C currently offers a choice of two formalisms for interface definition: the token syntax and associated `#pragma` extensions and the `tspec` syntax and tool.

The token syntax is stable, general, and well-validated. The current snapshot of ANDF uses it to specify all relevant API's: ANSI C, POSIX, XPG/3, System V.4. On the other hand, its syntax is somewhat clumsy, and in particular completely different from C syntax.

The `tspec` syntax is more recent and less general, since it is designed specifically for specifying traditional C interfaces. In most cases, `tspec` syntax is identical to C syntax, thus making it easier to convert existing specifications. Moreover, vanishingly few interfaces need the full generality of token syntax.

Barring unforseen difficulties, tspec appears to be the appropriate formalism for COSE interface definition.

Real COSE interfaces should be translated into ANDF

As of this writing, COSE header files have not been released.

When they are released, it would be useful to check that they can be converted to ANDF header files.

It would also be useful to have concrete examples of the difference between parameterized header files and platform-independent header files.

5. Conclusion: ANDF is the best way to specify COSE

ANDF can specify COSE interfaces precisely and flexibly.

ANDF tools can check that platform-dependent header files conform to COSE.

ANDF tools can check that applications conform to COSE interfaces.

The industry can only benefit from common interfaces if there are operational ways to validate correct use and implementation of these interfaces.

ANDF is thus appropriate technology for specifying COSE interfaces.