ANDF Features and Benefits

Dr. N. E. Peeling
TDF Project Manager
DRA Malvern. UK

1. Introduction

In mid 1991 the Open Software Foundation (OSF) announced that they had selected DRA's TDF as the base technology for their Architecture Neutral Distribution Format (ANDF). OSF is working closely with UNIX Systems Laboratories (USL) who have announced their intention to commercialize and distribute TDF so that Independent Software Vendors (ISVs) can produce a single version of their product in TDF which will then work on any UNIX SVR4.2 system, regardless of which processor SVR4.2 is running on. TDF is also a central part of a large European collaborative project called the Open Microprocessor systems Initiative (OMI) which is developing new European microprocessor systems (hardware and software). DRA and OSF are involved in a $14 million project in OMI which is dedicated to further developing TDF/ANDF.

ANDF (the concept) and TDF (the technology) have attracted significant support since OSF first announced their Request for Technology (RFT) for an ANDF. OSF's vision was to use ANDF to unlock the unfulfilled promise of Open Systems: that all Open System applications should run on all Open System platforms. The central idea is that the user could buy a shrink-wrapped application in ANDF form and install it on any compliant Open System platform.

ANDF is the most speculative technology that OSF has sought. The RFTs for Motif, DCE and DME have been technically challenging, but the desired properties of these technologies and their likely impact on the computing industry have been clear from the start. With ANDF the features and benefits have emerged over a period of time.

2. Features and Benefits for Software Developers

2.1 Features of ANDF

The central tenet of ANDF is that it should allow the creation of a large portfolio of shrink-wrapped applications in ANDF which can be moved from one architecture to another. ANDF is not just a distribution format. It offers features to help ISVs create new shrink-wrapped applications and allows existing semi-portable applications to be recast into shrink wrapped form.

Before describing what ANDF has to offer it is worth examining why porting existing applications is so time consuming.

Efficiency and functionality of applications are of great importance to ISVs. Writing strictly portable “lowest common denominator” code is usually not acceptable. What often happens is that the ISV creates a “mutable” application which through the use of (often complex) conditional compilation creates an efficient variant of the application for each target. This creates an elaborate set of headers which usually has to be modified each time a new target is considered. The modification of such headers is something of a black art because the headers do not define a clean portability interface (indeed it is difficult and sometimes impossible to construct such an interface within the confines of C). Once the headers have been modified, the application is compiled to native code for the new target, exposing the ISV to frequent problems from inconsistent interpretations of C and compiler bugs.

ANDF can represent an architecture neutral description of the application which abstracts a clean portability interface. This architecture neutral version can be shipped to the target where efficient architecture specific pieces of ANDF are substituted which satisfy the form and semantics of the portability interface. The substitution creates an architecture specific ANDF version of the application which is then translated into machine code.

DRA's TDF technology provides a C compilation environment which handles ANSI C (and other common dialects) with an extension called #pragma token, which allows the portability interface to be expressed in a single set of architecture neutral headers. This compilation environment need be the only compilation environment that the ISV uses. DRA is optimizing its design for the needs of an ISV creating shrink wrapped applications.

It must be stressed that ANDF is not a “magic bullet”. It does not, cannot, and should not mandate portability. This means that ANDF is not a magical solution to the problems of legacy software. You cannot just run a non-portable application through the TDF technology and expect to produce a portable ANDF version. However, ANDF allows an application to be recast into a shrink wrapped form. There is as yet limited experience as to how much effort a software developer should expect to have to expend in such a recasting. OSF's experience shrink wrapping Informix's Wingz (TM) product and DRA's experience shrink wrapping public domain applications suggests that the creation of a shrink-wrapped version of an application takes roughly the same amount of time as doing a port to a new platform. Once an application has been shrink-wrapped, its ability to run on a given platform is constrained only by the availability on that platform of implementations of all the portability interfaces which the application requires

2.2 Benefits of ANDF for all Software Developers

The benefits to software developers of using the TDF technology are primarily reductions in porting costs, but ANDF should also reduce testing costs. It is unlikely that when software developers start to use ANDF that they would be happy to ship a shrink-wrapped product without specific platform testing. However, testing costs are related to the number of problems encountered during testing, and an application shrink-wrapped using ANDF should fail such tests much less frequently - the whole shrink-wrapping process makes porting a much simpler process and the use of a single compiler front-end will reduce exposure to inconsistent interpretations of language semantics and compiler bugs.

Software developers will have a much freer choice of implementation language. Provided an appropriate producer exists for the chosen language the ANDF produced by it can be installed on any machine with a ANDF installer and the appropriate libraries. This means that a software developer is not dependent on compilers being available for the chosen implementation language on all the computers the application is intended to run on. To ISVs this may make it easier to move to C++. To certain users it may mean that special-purpose languages are less expensive to support (e.g. Ada for the defence community). To organizations with legacy software written in now unfashionable languages it may provide a means of supporting legacy languages at much lower cost.

At the current time the only compiler front-end for TDF is for C (ANSI and other dialects). A great deal of effort has gone into ensuring that ANDF can be used as an intermediate language in “conventional” compilers for languages other than C. Implementing the C front-end for TDF exposed a number of issues that related to the production of architecture-neutral ANDF. There are parts of the semantic definition of C, such as the precise definition of some integer promotions, which depend on the target. In addition, ANSI C was extended with the #pragma token facility to allow software developers access to TDF's features for describing portability interfaces. These issues have been satisfactorily resolved for C, but a similar exercise would have to undertaken for other languages if they are intended for use in producing shrink-wrapped applications.

ANDF may also reduce software developers' maintenance costs. For example it will be possible to have one port and product for a manufacturer's operating system that should work on all future releases of that operating system.

2.3 The ISV’s Perspective

The ISV can look forward to reductions in porting, testing and maintenance costs. This may allow ISVs to economically provide their products on more platforms and may reduce the time it takes to provide a product on a new platform.

2.4 The Users' Perspective

Many user organizations develop their own proprietary software within their own MIS departments. The MIS departments will derive the same benefits as the ISVs for their own software development and deployment within their corporate structure.

2.5 The System Vendors' Perspective

System Vendors are often large software developers and vendors to their clients. There are however additional benefits to System Vendors:

- applications in ANDF can easily be reinstalled. It is therefore less important to maintain binary compatibility in operating system updates. The key requirement will be for System Vendors to maintain the consistent implementation of portability interfaces.
- ANDF will insulate customers and ISVs from the precise definition of ABIs allowing ABIs to evolve more easily e.g. addition of superscaling or the move to 64 bit.
- ISV product availability in ANDF will make it easier to introduce a new architecture because customers demand application availability—System Vendors currently spend large amounts of money ensuring application availability on their platforms.

3. TDF Overview

The purpose of this section is to give a high-level technical overview of TDF. The section is included because the reader is probably curious about what TDF looks like. It also provides some substantiation for the remarks about portability made in earlier sections, as well as answering some common technical worries about ANDF—such as the issue of whether TDF will degrade the run-time performance of applications.

TDF represents an evolution of compiler intermediate languages. It fulfils some of the promise of earlier (failed) attempts to create a “universal” compiler intermediate language. In this scenario if there are n programming languages and m machines then instead of needing the product of *n* and *m* compilers one only needs the sum (*n*+*m*) “half compilers”. OSF refers to the *n* front-ends which compile the programming languages into ANDF as producers, and the *m* back-ends which translate ANDF into the machine code of the different machines are referred to as installers.

TDF is not a pseudo-code which represents machine instructions for some abstract stack or register computer. Instead it is a tree-structured intermediate language (IL) containing abstractions for common programming language concepts such a data structures, procedures, numbers, conditionals, loops, labels, jumps etc. These abstractions have been carefully designed so as to be able to accommodate the particular variants found in different programming languages. For distribution the TDF tree is “flattened” into a compactly encoded stream of bits. Although there have been language specific intermediate languages at the TDF level before (e.g. Diana for Ada), no-one before has attempted to design a multi-lingual IL at such a level.

In a producer which compiles from a programming language to TDF, very little information is lost. The only significant loss is syntactic sugar such as identifier names - which are kept separately for debugging purposes but need not be distributed with a shrink wrapped application. The aim was to keep information useful to code optimizations in the installers, but to lose information useful for reverse engineering. The syntactic sugar (which is the only information that cannot be recovered from a binary) is of no use to code optimizers and so is discarded.

The following subsections will examine different aspects of TDF - architecture neutrality; run-time performance of applications; ease of implementing new installers; and the representation of portability interfaces.

3.1 TDF: Architecture Neutrality / Run-time Performance of Applications

Because TDF abstracts programming languages it does not favor any particular architecture. The preservation of optimization information means that if a good compiler for C can be implemented for an architecture then a TDF installer can be implemented which provides as good performance when used as the back-end of the C to TDF producer.

TDF has a totally symbolic representation of memory which allows installers to choose a storage layout appropriate for the target. By contrast, most existing ILs assume a particular storage model and the front-end needs to be modified to generate different IL for any machine with a new storage layout.

3.2 TDF: Ease of Implementing New Installers

The TDF technology is designed to allow new high-performance installers to be implemented economically. To achieve this as many optimizations as possible, both universally applicable optimizations and machine specific ones, are carried out as TDF-to-TDF transformations. This means that TDF has to be able to represent the output of commonly used optimizations, which necessitated the provision of low level as well as high level features. This transformational software is portable and can provide as much as 70% of the code of a new installer. This installer technology has allowed DRA and OSF to provide installers for many of today's widely used microprocessor architectures (3/486, SPARC, MIPS, 680x0, VAX are complete with RS6000 in development).

As a measure of the effort to produce a new installer: Using DRA's model implementation as a basis, the SPARC installer was implemented by a DRA contractor to a level where it equals the performance of the best C compilers on a SUN SparcStation in approximately a staff-year. The implementation was performed by an experienced compiler writer who was familiar with SPARC but who had not worked with TDF before. The implementation was produced at a site remote from DRA with less than a week's consultancy from DRA.

3.3 TDF: Representation of Portability Interfaces

TDF contains a notion called “tokenization”. This is the ability of TDF to represent program as a TDF tree any part of which can be represented by a symbol called a “token”. The token can represent any part of the program detail (e.g. a type, a part of a type, a procedure, a macro, an expression or part of an expression, etc.) which is architecture specific. After distribution the token can have an appropriate architecture specific piece of TDF tree supplied as its definition. Token substitution produces an architecture specific TDF representation of the application which can be translated into the machine code of the target by the appropriate TDF installer.

Tokenization is the key to TDF's representation of the interfaces required for portability. These might be established APIs such as Posix, XPG3, SVID3, the AES etc. or they might be interfaces created by an ISV. An ISV's interface might abstract some low-level machine-specific manipulation which needs to be appropriately substituted on each target in order to achieve some added functionality or performance. An ISV's interface may also contain functions that are outside the coverage of widely available APIs. The ISV will then have to decide how to make implementations of token definition libraries, which implement the ISV's interface, available on all the different target platforms. It may be that ISVs sell implementations of their interfaces separately, or it may be that implementations for a wide variety of platforms are included with the shrink-wrapped applications.

The technique of token substitution has an impact on the relationship between the user, the ISV, the definers of APIs, and the implementors of APIs.

Consider the process of updating the implementations of interfaces. In the current software market the binary of an ISV's product will have bound into it a particular implementation of its interfaces, be that a particular version of X-Windows, or library calls from a particular version of an operating system (etc.). In the TDF world the application contains tokens for interfacing to system interfaces and during installation the user's environment supplies implementations that meet the interface standards expected by the application. At a later date the user can reinstall the application with later versions of (say) the operating system or X-Windows, provided the new implementations implement the same functional specification expected by the application.

The #pragma token extension to C can be used by API definers to produce machine-readable architecture neutral C headers for their standards. The C to TDF producer uses the information provided by the #pragma token syntax to do some validation checks on API implementations - these checks have already shown up a number of latent errors in existing API implementations.

It is impossible within an overview document such as this to give much more in the way of details as to how tokenization works. Other documents will give full details of tokenization and examples of its use.

4. Conclusions

TDF is the first technology specifically designed to support the shrink-wrapping of applications. This document has described how ISVs, users and System Vendors might benefit from the technology, and has related those benefits to particular features of the TDF technology. A number of other documents are being produced which will explore the TDF technology in greater detail.

For further information please contact:

Dr. Nic Peeling
internet: peeling%hermes.mod.uk@relay.mod.uk
janet: peeling@uk.mod.hermes
fax: +44 684 894303