EXPERIENCES WITH ECMA PCTE BINDINGS

John Dawes
ICL Secure Systems
Eskdale Road
Winnersh
Wokingham
Berkshire RG11 2SG
United Kingdom

e-mail : sjd@win.icl.co.uk

-- D R A F T --

1. INTRODUCTION

PCTE is a major focus of activity in the field of Public Tool Interfaces, and of its standardisation of ECMA was a significant event. The standardisation activity has several aspects of interest, notably the presentation as an abstract (i.e. programming-language-independent) specification and dependent bindings to programming languages (in this case Ada and C). The author believes that this process has broken new ground in the writing of specifications, and thrown up a number of new problems. As these are the problems that the publications of SC22/WG11 are intended to solve, our experiences in using those publications should be of interest.

I have been actively involved in the PCTE standardisation from the beginning, first as a member of the task group whose report laid the foundation for the standardisation, and then as Convenor of the task group TC33-TGEP that has carried it out. Throughout this paper 'I' means the author and 'we' means the members of TGEP, collectively or individually.

The views expressed in this paper are mine and not necessarily those of ECMA, TC33, TGEP, or ICL.

2. A BRIEF HISTORY OF PCTE

PCTE is a public tool interface, i.e. it is an interface to a set of facilities on which environments to support systems engineering projects may be constructed out of integrated tools. The main facilities supported by PCTE are: an object base (or data repository) based on an entity-relationship model; a schema management system processes, with facilities for interprocess communication; transaction and locking facilities for concurrency and integrity control and mandatory and discretionary security control.

PCTE started life in ESPRIT project 32, "A Basis for a Portable Common Tool Environemnt", which produced a specification in C, an initial implementation, and some tools. The specification went through a number of versions, ending in version 1.4 [PCTE86]. The CEC later commissioned an Ada version [PCTE87] and a formal definition using an extension of VDM [VIPK88].

In order to maintain and exploit PCTE, the CEC established the PCTE Interface Management Board (PIMB) in 1986. The PIMB published a revision known as PCTE 1.5, and then established an ad hoc task group to consider the standardisation of PCTE. This group reported in June 1988 [ASTF88], strongly recommending that the standard should be an abstract (language-independent) specification and separate dependent language bindings.

In parallel, in 1986 several nations of the Independent European Programme Group (an independent grouping of the European members of NATO) embarked on a collaborative programme to enhance PCTE to

Upon request from the PIMB, ECMA undertook the standardisation of PCTE, and formed a Technical Committee TC33 in February 1988 with this objective. [ASTF88] was accepted by TC33, and a task group (Task Group for ECMA PCTE, TGEP) was formed in November 1988 to produce the Abstract Specification and bindings for Ada and C. The basis for ECMA PCTE was to be PCTE+ issue 3, both Ada and C versions.

The Abstract Specification was accepted by the General Assembly of December 1990 and published as Standard ECMA-149. The C Binding was accepted by the General Assembly of June 1991 and published as Standard ECMA-158. An Ada Binding is scheduled for consideration by the ECMA General Assembly of December 1991, and work has begun on a C++ binding.

3. THE FORM OF THE ABSTRACT SPECIFICATION

Although the basis for the standardisation was the C and Ada version of PCTE+ issue 3, the task group was also charged with considering all relevant technical concerns raised by members. The consequence was that the task group's efforts could not be spent wholely on converting PCTE+ issue 3 into an abstract specification plus bindings; many proposed changes of functionality had to be considered also. The practical outcome was that the Abstract Specification, which acted as the vehicle for recording decisions on these questions of functionality and as the medium for conducting the discussions, was fairly well on the way to completion before serious consideration was given to problems of producing the bindings. Some of the consequences are described below.

[ASTF88] distinguished between formality and abstractness of a specification such as PCTE: formality is to do with rigour and precision, whereas abstractness is to do with freedom from implementation-dependent detail. [ASTF88] recommended an informal abstract specification as the PCTE standard, i.e. one expressed largely in natural English, at a level of abstraction above the details of the programming language bindings, for the following reasons.

1. To provide a single authoritative standard.
2. To provide a reference source for future language bindings.
3. To reduce duplication of work between the various language bindings.
4. To provide a firm basis for considering the consistency of different bindings.
5. To conform to the preferred approach of standardisation bodies.

These reasons have been largely vindicated in practice. 1) is hardly disputable. 2) will be demonstrated when a new binding is produced; preliminary work has begun on a C++ binding, and it appears that few or no changes are needed to the Abstract Specification to allow such a binding to make full use of the object-orientatedness of C++. 3) and 4) have been borne out by the work on the C and Ada bindings, which were developed in parallel with each other and the Abstract Specification. Although they revealed some problems in the gap between the assumptions made by the Abstract Specification and the abstraction level of the bindings (see below), both bindings took a fraction of the time to produce that would have been required without the Abstract Specification. It can also now be demonstrated that the C and Ada bindings (and any future bindings) are consistent in a well-defined sense, both being related in defined ways to the same specification, this consistency being guaranteed by the conformance requirement on bindings. 5) was perhaps rather anticipatory at the time (June 1988) but it is now accepted practice.

The ASTF decision to advise against a formal definition was not unanimous. A minority (including the author) believed that not only was a formal definition possible and desirable but that it might well take less time to produce than an informal one, and could then be the basis of an informal translation of high quality; but this minority failed to carry the day, so the standard is not formal. However, partly as a result of a presentation of the results of the VIP project [VIPK88], which impressed the members of TGEP a good deal, formal methods were used to some extent: VDM notation (VDM-SL) was used to define the state, some data types, and the operation signatures.

Had TGEP been more aware of CLIDT and CLIPCM, and had those documents been in a stabler state, then

4. THE BINDINGS

4.1 Background

Although the PCTE Abstract Specification is pioneering effort in some ways, there is a good deal of activity at present in this field, in particular in the Bindings working group ISO/IEC/JTC 1/SC22/WG11, and TGEP tried to take full advantage of their work. Two documents in particular were relevant: the Guidelines [DTR10182] and the Common Language-Independent Datatypes [CLID90].

Writing the Ada and C bindings started at about the same time, but the Ada bindings lagged behind due to shortage of effort. However there was enough overlap so that a common approach could be worked out and lessons learned from each binding could be fed back to the other. As the C binding is now published, most of my examples are drawn from that binding.

The approach taken was based on the idea that the Binding standards would be completely dependent on the Abstract Specification; no attempt would be made to make them self contained. This was an application of the excellent rule of never to say anything twice (because the changes of saying it differently are overwhelming). Each binding document was to contain only the specification of the interface in the binding language, and the mapping between it and the abstract specification. In practice this rule has been bent in a number of places, and information has been repeated from the Abstract Specification; notably the operation signatures, to make reference easier.

The method producing a binding was in three stages.

1. A number of working papers were produced dealing with questions of strategy. These were designed as issued affecting the binding as a whole; a few were obvious at the outset such as; how data types and operations should be mapped, what the overall structures of the interface should be and how errors should be handled.
2. Once the strategic questions had been at least provisionally settled, the next stage was to define the mappings from the data types of the Abstract Specification to the binding language. This proved to be the hardest part of the binding exercise, causing the most heated arguments, and requiring many iterations before solutions were received that were acceptable to TGGP. The CLIDT was used extensively in this stage for both C and Ada bindings; this is discussed below.
3. The final stage was the easiest, though not quite so straightforward and mechanical as we had hoped; this was to write the operation specifications. Some problems were particular to the operations, for example dealing with optional parameters; others caused the data type bindings to be revisited. The CLIPCM was not considered for use at this stage as there seemed to be no conceptual difficulty in mapping VDM operations directly to C functions or Ada subprograms in a systematic way, and so no need for an intermediate level. Also, it seems aimed more a providing mechanism for mixed-language programming than an abstraction of the normal procedure calls of programming languages.

4.2 Use of the Guidelines

The Guidelines were used at the beginning of stage (1) of the Ada and C Bindings, and provided a valuable checklist of strategic issues and source of suggested solutions. Some comments arising from this experience [???] were submitted to WG11; they were positively received and are not repeated here.

The first point considered was the choice of functional binding methods in Chapter 2. The relevant functional binding method for us was clearly Method 1: provide a completely defined procedural interface. However the existence of Method 5: binding pre-existing language elements, brought to light the issue of what to do about the functional facilities in the binding language that overlapped facilities provided by PCTE; a typical example being the binding language input-output facilities. There seemed to be no ideal answer to this. In the end it was decided to provide PCTE-defined input-output and to leave the effect of using the binding language input-output to be defined by the implementation.

Guidelines 2 and 3 : Either the language or the system facility committee should have primary responsibility, and should consult the other as early as possible. Accepted.

Clearly the system facility committee (TGEP) had primary responsibility in this case. Guideline 3 has not really been put into practice yet. In fact liaison with the JTC1 language committees for the binding languages has been informal and one-way: members of TGEP have sat on both the Ada and the C Working Groups of SC22, but the PCTE Bindings have never been considered by those Working Groups. This is a concern that is being addressed now as part of the international standardization process for PCTE.

Guideline 4 : Specific guidelines should be produced for bindings of particular system facilities. Accepted. The guidelines for PCTE Bindings have not yet been written down coherently, but again we hope to do so as part of the Rationale. No language-specific guidelines have been seen, though it was believed that WG9 atone time intended to do something for Ada.

Guideline 5 : The binding standardization should not get ahead of the system facility or language standardization. Accepted. The choice of binding languages was out of the hand of TGEP, as the decision was made by the parent body, TC33, on the base of user need (issue 8, alternative 3). TGEP incline to alternative 1 (only standardized languages), but in practice settle for alternative 2 (also languages being standardized). The C Language Standard was greeted with relief, and the lack of a C++ Language Standard is causing concern.

Guidelines 6 and 7 : Different language bindings should not differ unnecessarily. Accepted; in fact we found that the C and Ada bindings we produced were far more similar in structure than the original C and Ada versions of PCTE+ (where the Ada version was produced from the C version, but is self-contained). I believe this is because we started from an appropriate level of abstraction, so that the structure of the specification was not distorted by inessential details of the language; the two languages (which from an abstract enough point of view are not all that different) could then be applied in a uniform way.

An example is the mapping of operations that return sequences of unknown length. In PCTE+, the C version requires the caller to set up an array into which the operation returns as many values as it can, indicating if there are more values to return; while the Ada version requires the caller to acquire an iterator, i.e. a value of a private type which is passed to the operation which returns one value at a time. Both methods could be (and were) defended as being in accord with the customary practice of the language, but there appears to be no technical reason why Ada and C should use different methods, and probably the same method will be used for both ECMA PCTE bindings.

Guidelines 18, 19 and 42 : Lists of abbreviations of function names, and guidelines for choosing one, are needed. Rejected, but not without some difficulty. Many of the names in the Abstract Specification are fairly long, and there was uncertainty about the length of identifiers allowed in C. It was decided that a list of abbreviations to 29 characters was needed. Two rival lists were produced, one based on what its author considered were the normal rules of abbreviation in English, the other (totally different) on what its author considered were the normal rules of abbreviation in programming. Neither was based on omitting redundant words (guideline 42). Luckily, protracted discussion was avoided by the realisation that standard C does not place a limit on the lengths of identifiers, providing the first 31 characters are unique (except for a concession to obsolescent compilers). It was ensured that all names were unique in the first 31 characters and both lists of abbreviations were archived.

Guidelines 28, 29 and 30 : A generic binding may be provided. Not applicable. It is hard to imagine how a generic binding could be prepared even for such similar languages as Ada and C, though some features of all bindings are defined in a special clause of ECMA-149; these include the mapping of PCTE datatypes to CLI datatypes, described below.

Guideline 32 : The binding should take account of likely language-processor limits. Accepted up to a point. This was a live issue for the C Binding, as some existing implementations have limits which are disallowed or deprecated in the Language Standard (e.g. on identifier lengths). We certainly did not want more than one Binding (issue 9, alternative 2); in practice we tried to take some account of known

Guideline 33 : If error numbering is used, common errors and each set of language-specific errors need disjoint sets of numbers. Accepted with modification, as ECMA PCTE does not use error numbering, but still needs to keep common and language-specific errors separate.

Guideline 35 : Parameter order should be preserved. Accepted for the C Binding, but under discussion for the Ada Binding where there are reasons for varying the order (to allow defaulted parameters to be omitted with positional association).

Guideline 36 : Parameter direction (in/out) should be preserved. Accepted, but difficult to implement in C which has no definite concept of parameter direction.

Guideline 48 : All binding documents should use a common schema based on the functional standard. Accepted, although we rejected it at first on the grounds that the appropriate structure for a flat language like C was inappropriate for a structured language like Ada. Indeed the PCTE+ Ada version is a highly structured document, following the nested package structure in the nested section structure of the document. However, when it came down to it, we all agreed that for ease of reference a flat structure (exactly following the flat structure of the Abstract Specification: one clause per operation with a standard layout) was the best, and could as easily be achieved for Ada as for C.

Of course, unlike the ECMA PCTE Bindings, the PCTE+ specifications are self-contained documents and have to define the semantics as well.

5. DATATYPE MAPPING AND USE OF CLIDT.

5.1 General.

The other document from WG11 that gave us a lot of help was the Common Language-Independent Datatypes, CLIDT. This gives a definition with a strong mathematical flavour of a set of primitive datatypes and constructors for defining further datatypes, and appeared to solve the problem of defining a satisfactory mapping from the datatypes defined in the Abstract Specification (called "PCTE datatypes") to those used in the Bindings (called "language datatypes"). This is done in two stages : first the PCTE datatypes are mapped to a selection of CLI datatypes, and then those CLI datatypes are mapped to the binding language datatypes.

By using CLI datatypes as an intermediate stage, it was hoped that for most PCTE datatypes the first stage need be done only once. As it turned out, the Abstract Specification was published before the mappings could be worked out in enough detail to give us confidence that that could be done, so ECMA-149 contains mappings to CLI datatypes only for basic PCTE datatypes. It seems likely now that a generic mapping CLI datatypes would be possible for most but not all PCTE datatypes, the main exceptions (discussed below) being sets and optional types.

The conformance clause of ECMA-149 for Bindings demands a mapping from PCTE datatypes to the language binding datatypes, but the mapping need not be one-to-one : different PCTE datatypes may be mapped to the same language datatype, and subtypes of one PCTE datatype may be matched to different language datatypes. Both these concessions were required.

The PCTE datatypes that need to be mapped are those used as operation parameter and result types : as noted, they are defined in VDM-SL in the Abstract Specification. They fall into 3 classes, which can be considered separately : basic datatypes, designator datatypes, and constructed datatypes.

5.2 Mapping Basic PCTE Datatypes.

Basic PCTE datatypes are the basic types of VDM : Boolean, natural, integer, real, and enumeration types: time (which is not a VDM type) is also included. These present few problems; the mapping to primitive CLI types is straightforward, and is contained in a separate clause of ECMA-149. One complication is that the Standard allows implementation-defined limits to be imposed on certain quantities, such as the range of integers supported, but defines bounds on those limits. This is easily accommodated by using CLI

Enumerated types cause a particular problem. In VDM, an enumerated type definition such as

T1 = RED | BLUE | YELLOW

defines T1 to be the union of 3 types, each containing a single value. It follows that if another type overlaps T1 :

T2 = RED | AMBER | GREEN

then the value RED is common to both types. It is unclear if this is true of CLI Enumerated (or State, which we probably should have used) datatypes or not; it is certainly not true of C or Ada enumerated typed. (In C all enumeration value identifiers must be different; in Ada the same identifier can be used in two enumeration types but represents different values.) We therefore had to make a number of changes in the enumeration types in the Bindings, combining them where common names were clearly meant to denote the same values, and (for C) changing some names for uniqueness. For simplicity, and because the latter change was needed for only one binding, we made these changes on the mapping from CLI datatypes to language datatypes, keeping the PCTE TO CLI datatypes one-to-one for enumerated datatypes, and assuming that IDN follows VDM-SL rather than C or Ada as regards names of enumeration literals. (One of the advantages of an informally defined notation such as IDN is that it is possible to fudge issues such as this. VDM-SL is formally defined and such issues cannot be dodged.)

The mapping of an enumerated datatype to C is further complicated if the set type of the enumerated datatype is also required; indeed there are 3 mappings, according as the enumerated type, the set type, or both are required. This is a language-specific technicality caused by the nature of C's type declarations.

5.3 Mapping designator PCTE datatypes.

Designators are undefined values used in the Abstract Specification to refer to entities in the PCTE object base - objects, links, attributes, and their types. As described above, they are idealized references, and the mapping to CLI datatypes must take account of the process of accessing an entity from an actual reference. One aspect of this is that an actual reference might not refer to an entity at all due to an error that the accessing process must detect. Another aspect is that in some cases designators represent more than one kind of actual references, for use in different situations, and that had to be described somehow in the Standard.

The solution adopted is to introduce a new set of datatypes, corresponding to designators at a low abstraction level, unimaginatively called references. They are defined either as abstract datatypes (object references) or as strings with a certain internal syntax (the others). Operations are defined for creating and handling references, and the process of accessing an entity from a reference is defined. The mapping to the binding is then in terms of references and designators are not mentioned.

In fact, link, attribute, and type reference datatypes are strings and are mapped in the same way as other strings (via the CLI datatype character-string). The object reference datatype is mapped to a CLI private datatype and from there to a C opaque (word printer) type or an Ada private type.

5.4 Mapping constructed PCTE datatypes.

Constructed PCTE datatypes are those constructed by use of the VDM type constructors. The most used type constructors in the Abstract Specification are sequence, set, union, and option; there are also some uses of record and map.

Each type constructor raised particular problems, which are described below, but there is also a general issue of the broad approach to be taken to their mapping. One approach is to map each VDM-SL type constructor to a CLI datatype generator, defined in the CLIDT or specially by a datatype generator

and possible inefficiency into the use of such types, which is avoided by the other approach of mapping; choose the most appropriate language datatype in each case and map to that. The disadvantage of that is that all the implementation details have to be defined (or explicitly said to be implementation-defined) which is tedious, messy, and out of place.

Sequences are an example. hey could be mapped to C sequences or Ada arrays, but only at the cost of solving the problems of returning variable-length sequences from operations, which is really an implementation design problem. Accordingly the VDM-SL sequence type constructor is mapped to a CLI datatype generator Sequence, and all CLI sequence datatypes are mapped to an opaque C datatype Pcte-sequence.

Sequence is derived from List, with a few more characterizing operations such as indexing, which it is felt every binding should support. Pcte-sequence is supported by an enumeration datatype representing all possible element types.

However there is an exception to this mapping. Strings in VDM are just sequences of characters, but text strings are handled specially in most programming languages, including Ada and C, and it would be outside the spirit of the language to ignore these. Strings are used in the Abstract Specification for various purposes, and analysis showed that they could be classified in two ways:

— the repertoire of characters is either the complete 8- bit character set, or else a subset of the ASC11 graphic set;

— the length is either limited to a fairly small maximum, or is unlimited or with a large upper bound.

Most cases were limited repertoire and length, or unlimited repertoire and length, and the few other cases could be treated as one or the other. Accordingly two mappings of strings are defined : the first case (called bounded strings) being mapped to the binding language native string type, and the second (unbounded strings) as for any other sequence.

For no very good reason the split occurs in the mapping from CLI to language datatypes, the PCTE datatype string being mapped to the CLI datatype character string with a repertoire indicator intended to indicate the entire 8-bit character set.

Sets are used a good deal in the Abstract Specification, and offer another good example of an abstraction covering two different situations at a concrete level. Sets are handled in two different ways in normal programming practice. Sets of elements from a small universe of values, e.g. an emuneration type, are often represented by fixed-length bit arrays, with each bit position indicating the presence of absence of a particular element. Sets of elements from a large universe of values, e.g. integers or records, are generally represented by sequences of element values, possibly with a canonical ordering and with an appropriate convention regarding repeated elements.

Set types of both kinds are used in PCTE, and it was decided to follow custom and map them differently. This time, however, the split occurs in the PCTE to CLI datatype mapping, the reason being that two different datatype generators were wanted. The VDM set type constructor for the first kind of set type (called bounded set types) is mapped to a CLI datatype generator Bounded-set, derived from Set by restricting the cardinality of the elements. Each CLI datatype of the bounded set family is then mapped to a fixed- length bit array (in C, to an integer with an enumeration type to name the bit positions).

The VDM set type constructor for the second kind if set type (unbounded set types) is mapped to the Sequence datatype generator; to take care of set operation (including set equality) an additional characterizing operation Normalize is defined for Sequence datatypes, which removes repeated elements and reorders the sequence into an implementation- defined canonical order.

Record types are used in a few places, but present no particular problems; they map straightforwardly via CLI record datatypes to C structures and Ada records. There are no complications of discriminants or variants in VDM-SL. Cartesran product types are used occasionally, and are treated as record types with no

an even smaller range (of only 4 values); the other is like an unbounded set type, with large domain and range. The bounded map type could be conveniently and ingeniously mapped to pair of bounded sets, while the unbounded map type needed to be mapped as an unbounded set of pairs of values representing the maplets. A full mapping definition was devised, from the VDM map type constructor to a new CLI datatype generator Map (derived from Table) and thence to the two C datatypes, complete with alternative code fragments for the characterizing operations. However pressure of publication of the C Binding caused it to be dropped, and all that remains is a mention of an undefined CLI datatype generator Map, and a direct mapping to C of each of the PCTE map types.

The union datatype generator is easily handled. Unions of VDM enumeration datatypes map to CLI enumerated datatypes as described above; otherwise the union type constructor is mapped to the CLI datatype generator Choice, and each datatype of the family is mapped to an appropriate union datatype (C) or discriminated record type (Ada). This is another case where to map all the datatypes of the family to a single abstract language datatype seemed to ignore language facilities provided exactly for the purpose in hand.

Optional types, on the other hand, presented a problem. An optional type such as [T12] in VDM represents the union of type T1 with the nil type, which has the single value nil; this is used conventionally to represent the absence of a value. It would be possible to map the VDM optional type constructor [] to the CLI Datatype generator Choice using (I think) the CLI datatype Null to represent the VDM nil datatype. This could then be mapped, as described above for the general Choice datatypes. However, this again seemed to ignore specific language features (such as NULL pointers in C and null access values in Ada) provided for the purpose, and to introduce unnecessary overheads in many cases. Analysis of the use of optional types in the Abstract Specification showed there were 4 kinds of cases to deal with.

1. In some cases an optional parameter type was used to indicate a default value which could be supplied. This kind of optionality is not mapped to C, but a note is given in the C Binding that the effect of not supplying the value is got by supplying the default value. In the Ada Binding, the optionality can be indicated by supplying a default value for the parameter; this raises the issue of Guideline as to whether such Ada parameters should be moved to the end of the parameter list.
2. In other cases optional parameters were used because an operation had several functions, some of which did not need the parameters. In these cases the abstract operations are mapped to several binding operations, one per function.
3. In many of the remaining cases, an optional parameter or result type is mapped to a type with a natural default value in the language; e.g. C pointer value NULL or Ada null access value. These cases are mapped in the natural way.
4. In the few remaining cases, the fallback mapping of a discriminated record with a special nil value is used.

6. COMPLIANCE OF ECMA PCTE TO THE CLIDT

It is difficult to see how to apply the compliance clause of CLIDT to ECMA PCTE, which may indicate that we have not used it exactly in the manner intended.

Ideally, perhaps, the Abstract Specification would define PCTE datatypes in CLIDT IDN instead of VDM-SL and there would be standard mappings from CLI datatypes to all languages; a large part of the mapping would then have been done. As it is, it seems clear that neither the Abstract Specification nor the Bindings can comply directly, since they use datatypes not defined using CLIDT. Nor can they claim indirect compliance, since the Abstract Specification defines only outward mappings and the bindings define inward mappings, whereas the CLIDT requires both. And even partial compliance is beyond us, as we have no use for Array.

The conclusion is perhaps that PCTE is not the sort of thing that can be compliant to the CLIDT; and yet language bindings for abstract specification is surely exactly what CLIDT is aimed at.

We must give serious consideration to establishing a complete standard mapping of PCTE datatypes to CLI datatypes. This would certainly require some changes to the interface, and is impossible until CLIDT is itself standardised. It is probably too late to convert the Abstract Specification to use CLIDT IDN instead of VDM-SL for defining datatypes; at least until the next major version.

We may need a standard mapping from abstract operations to CLIPCM procedure signatures, though this is unclear. If the Abstract Specification is changed to use IDN for defining datatypes, then it should also use IDN for defining operation signatures, for consistency (and the state, which is a collection of objects).

The whole exercise has certainly benefited from the fact that the Abstract Specification did not get too far ahead of the Bindings - in fact it is expected that future bindings will continue to turn up improvements to the Abstract Specification, though diminishing with each successive binding.

7.2 LESSONS FOR WG11

I thing the work reported here is one of the first efforts to use the WG11 products seriously. It would have benefited from a clearer idea of how they were to be used, particularly CLIDT and (perhaps) CLIPCM. This seems to me to indicate that WG11 should pursue a number of objectives; they compete for resources, but I have not tried to prioritise them.

— To progress the standardisation of the documents as rapidly as possible we used versions 3,4, and 5 of CLIDT; though each improved on its successor, it would have been of more benefit to us if version 3 had been frozen. This is not to say that versions 4 and 5 should not have been produced, rather that the point of diminishing returns should be recognised.

— To get the documents into real use and encourage reporting back experiences. This is not easy, especially to persuade users to invest the effort required to report back experiences.

— To give some thought to the problem of how the documents are to be used. I hope this paper has shown that this is not obvious, at least as shown by the problems of defining PCTE's compliance. It could be solved by writing a User's Guide, but perhaps it would be better to include it in the Guidelines.

PCTE datatype		CLI datatype		C datatype
Base:				
Boolean	--->	Boolean	--->	Pcte_boolean
natural	--->	Range of Integer	--->	Pcte_natural
integer	--->	(Range of Integer)	--->	Pcte_integer
real	--->	Real ( )	--->	Pcte_float
time	--->	Date-and-Time ( )	--->	Pcte_time
enumerated	--->	Enumerated (State)	--->	enum
Constructed:				
seq of T	---> unbounded	Sequence of T (From List)	--->	Pcte_sequence
set of T	---> bounded	Bounded-set of T (From Set)	--->	Pcte natural

T1 x T2X...				unbounded Pcte_sequence
map T1 to T2	--->	(Map of (T1,T2)) (from Table)	--->	(of maplets) pair of Pcte_natural
string	--->	character_string ("all")	unbounded ---> bounded	Pcte sequence (of char) string
T1/T2/...	--->	Choice of (T1,T2...)	--->	various (T1,T2..)
[T1]	--->	(no CLI datatype)	--->	various mappings
Designator:				
object reference	--->	private	--->	word *
link ref. } attribute ref. } type ref. }	--->	character-string ("all")	--->	string

Figure 1: Two-stage mapping of PCTE datatypes

References

[ASTF88] Report to the PCTE Interface Management Board of the Task Force on Abstract Specification, 24th June 1988. (PIMB/ASTF/REP/01,02,03.)

[BGMT89] G. Boudier, F Gallo, R Minot, I Thomas. An Overview of PCTE and PCTE+ ACM SIGPLAN Notices, Vol.24, No.2, February 1989.

[CLID90] ISO/IEC JTC1/SC22/WG11/N190 Common Language-Independent Datatypes, Working Draft No. 4, 6 September 1990/

[DaDa89] H F Davis and S H Dawes. ECMA PCTE: Formalising an Interface Definition. Ada UK 8th International Conference, September 1989 (Ada User volume 10 Supplement.

[DTR10182] ISO/IEC/DTR 10182 Guidelines for Language Bindings (6 February 1990).

[GP188a] The German PCTE Initiative. Requirements for the Enhancement of PCTE/OMS, version 2.0, 26 September 1988.

[GP188b] Introduction to the Specification of the GPI OMS Data Model, version 1.0, 16 December 1988.

[PCTE+a] PCTE+, C, Functional Specification, Issue 3, 28 October 1988.

[PCTE+b] PCTE+, Ada, Functional Specification, Issue 3, 28 October 1988.

[PCTE86] PCTE, A Basis for a Portable Common Tool Environment, Functional Specifications, Fourth Edition, November 1986.

[PCTE87] PCTE, A Basis for a Portable Common Tool Environment, Ada, Functional Specification, First Edition, June 1987.

[PCTE89] IEPG TA13. Introducing PCTE+, April 1989.

[RATO3] PCTE+/RAT/03 Rationale for the Changes between the PCTE+ Specifications Issue 3 dated 28 October 1988 and the PCTE Specifications Version 1.5 dated 15 November 1988 (Issue 3, 6 January 1989).

[VIPK88] VIP Kernel Interface: Final Specification, December 1988.

individuals who have contributed to previous PCTE and PCTE+ specifications.

Some of the material in this paper appeared in a paper [DaDa91] by me and H.F.Davis presented at the Software Engineering Environments Workshop SEE'91 at UCW Aberystwyth, and appears by permission of H.F.Davis and of F.Long, the Editor of the SEE'91 Proceedings.