4/19/91

From: Ed Greengrass
To: CLIDT-ers, CLIPCM-ers
Subject: Mapping, Marshalling

What follows is yet another Barkmeyer-Greengrass dialogue on the related subjects of mapping and marshalling. It winds up in agreement. Does everybody (anybody?) else agree?

From: edbark@cme.nist.gov (Ed Barkmeyer)
To: Ed Greengrass
Subject: Re: Marshalling, CLIPCM

"Marshalling" was a term I had heard once before 89, when we got serious about CLIPC and RPC. It seems to mean "the process by which the actual parameters to a procedure call are made available to the called procedure itself, or by extension, to the standard interface." This process usually involves ordering and locating conventions and may require creation of hidden parameters, copying and/or conversion of values, etc. The RPC troops invented the term "unmarshalling" to refer to the inverse process of making the actual parameters of the standard interface available in the particular interface (form) required by the language. I believe that such a definition either is or should be a part of the CLIPC.

What has happened in the last two meetings is that I, and I think others, but not Brian Meek, have come to accept Craig's view of the relationship of mappings to the CLIPC. No consensus on this has ever been stated in either WG11 or X3T2, but here goes: The "outward mapping" from the programming language to the CLID is for reference semantics only and HAS NO BEARING ON THE CLIPC or RPC WHATSOEVER.

The "inward mapping" from the CLID to the language defines a "standard means of support" in the language for each CLI datatype. (There may be some that the language cannot reasonably support at all.) There is NO requirement for the "inward mapping" to have a "unique inverse". That is, more than one CLI datatype can be mapped into the same language datatype or a subtype.

The inward mapping should CONTAIN the inverse of the outward mapping. That is, if the outward mapping OM maps language type T to CLI type X, then the inward mapping should map X into (onto?) T. But even that might be a problem. Consider languages which have multiple Integer datatypes, e.g. Fortran INTEGER*k. INTEGER*2 and INTEGER*4 will map onto different Ranges of Integer, but different Ranges of Integer are not distinct datatypes - they are POSSIBLY DISTINCT subtypes of Integer (and they may overlap). The inward map for Integer and Ranges thereof may or may not map anything into INTEGER*2, particularly since it is unlikely that either the CLIPC or RPC will want to distinguish Range of Integer from Integer.

On the other hand, if the inward mapping for language FRIML maps CLI datatype D onto language datatype E, then there is a "local inverse" at each calling parameter. That is, if the IDN defines Procedure SNARK([IN] boojum: D), then the CLIPC marshalling has a right to expect that the FRIML procedure prototype would look like SUBROUTINE SNARK(boojum: E). This implies that the standard marshalling routines for FRIML calls will include the "conversion" of values of FRIML datatype E to interchange (CLI) datatype D (to be used when the target is D!) and the standard unmarshalling routines will convert values of CLI datatype D to FRIML datatype E. (What FRIML datatype E is mapped to by the outward mapping is irrelevant.) The reason this works is that the CLIPC and RPC operations are steered by the IDN and thus by the CLI datatypes, not by the language datatypes. The inward mapping defines two sets of conversion routines - the marshalling routines and the unmarshalling routines - and the actual

interface selects a conversion routine based on the direction of the call and the CLI datatype involved.

Now to the example from my and your contribution. The FORTRAN inward mapping must define ENUMERATED to map to INTEGER *AND* the first enumerated value to map to 1, the second to 2, etc. If the FORTRAN programmer wants to use 4 thru 7, then he is not using the standard mapping. He can define any mapping he pleases, as long as:

1) he defines that mapping somehow (and that is what the CLID does not currently define); and

2) he provides the necessary conversion routines; and

3) he declares the use of those routines for that CLI datatype (or for specific parameters of specific procedures) in the IDN.

What is unresolved here is what exactly the CLIPC/RPC service provider needs to know about the user-specific mapping, other than the conversion routine names. The problem is that for each routine, one of the parameter and result has a CLI datatype, but the other has a language-dependent datatype whose syntax, and potentially representation, are only meaningful in that language. So, for example, a FORTRAN preprocessor can look at the IDN and figure out what the FORTRAN procedure prototype is, when only the standard mappings are used; but if the user overrides the standard mapping, he has to have some way to tell the preprocessor what the FORTRAN procedure prototype should be, and what the procedure prototype for the conversion routine is. And to make the override capability portable, the means of such specification has to be standardized, either in the IDN or in the FORTRAN specialization, wherever that is.

It is this whole notion of allowing the user to specify what he wants that requires some kind of formalism for mappings, and this has not been addressed by any of the CLID, CLIPC or RPC deliberations to date.

3/20/91

To: Ed Barkmeyer
From: Ed Greengrass
Subject: Marshalling, Mappings

Dear Ed

I've read your last letter several times in an effort to cudgel my still foggy brain. It seems to me that a point you made in your response to Craig needs to be repeated, expanded and waved about (a task I may shortly undertake). Data exchange is only one of the goals of CLIDT and CLIPCM/RPC is only one (admittedly important) instance of data exchange. It should not dictate the thinking and terminology of CLIDT though it can certainly influence it.

Let me back up for a moment. The starting point for data interchange is that we have program P1 in language L1 exchanging data with program P2 written in L2 using an encoding of CLIDT as the exchange medium. Hence, we have an (dare I say it?) outward mapping to CLIDT and an inward mapping from CLIDT. The key question is who defines these mappings and for what purpose. My impression is that when people speak about THE inward mapping, what they mean is the mapping to L2 DEFINED BY THE L2 LANGUAGE COMMUNITY. But mappings may also be defined by CLIPCM, RPC, service communities (e.g., database), application domains, etc. Now, let me cast what I think Craig is saying (or what you believe Craig is saying) in my CLIDT terms.

Procedure calls are an example of a client-server relationship. The server advertises how he wants to be called, an advertisement that is partly in CLIDT. When you say that "the CLIPC and RPC operations are steered by the IDN and thus by the CLI datatypes", what you mean (I think) is that they are steered by

this advertisement. Since the advertiser of a procedure presumably chooses CLIDT types that fit the native types of his procedure (written say in L2), the inward mapping is no problem. The procedure call has to conform to this advertisement and moreover may be written in L1 where L1 may be != L2. Hence, the outward mapping may be problematic (even impossible!) and how to handle it may have to be dictated for L1 by the CLIPCM/RPC community, the L1 community or (quite possibly) by a joint CLIPCM/RPC/L1 working group. Somebody chooses to call an outward mapping defined by such a community "marshalling" (or perhaps this outward mapping is only part of marshalling). CLIPCM and RPC are certainly entitled to define marshalling any way they like and to include an outward mapping in it, but this is only a special case of CLIDT outward mapping.

On the other hand, CLIPCM has problems of its own of which CLIDT knows nothing. To accommodate these problems, CLIPCM may have to specify CLIPCM-specific exchange, encoding and conversion rules. Hence marshalling may well contain more than a CLIDT outward mapping but the same thing applies to the unmarshalling. Hence, it would be a mistake to assume that once the procedure call has reached its "target" in correct CLIDT form, the rest is just a standard inward mapping (although that is a crucial PART of the unmarshalling).

Now what about the concept that the inverse of an inward mapping doesn't have to be unique? Specifically, what about the case where CLIDT types D1 and D2 both map into the same internal type E in language FRIML. Whether that is a reasonable thing to do depends not only on FRIML but also on the community that is defining the mapping. Several cases seem to arise here:

(1) D1 is definable in terms of D2. Since I am not a minimalist, I see no objection to this case and it raises no problems for the inverse mapping.

(2) D1 contains a property that D2 does not but not vice versa, e.g., Enumerated is State with ordering added. In that case, D1 and D2 can be mapped into E which corresponds to D1 but not into F which corresponds to D2. That is, we can map both State and Enumerated into an internal Enumerated type without loss but not into an internal State type. But what if the mapping is being defined for a community that has much need for a State type but little or no need for an ordered State type? In that case the mapping into State may be quite acceptable. However, a pragma should specify what is being lost. More than that, it might specify an extra-CLIDT convention to be used when an ordered State is needed, e.g., suffix each state value by an integer that specifies its ordinal position, e.g., PHOOEY1, BLAT2, YUCK3, etc.

(3) D1 contains a property that D2 does not and vice versa, i.e., D1 and D2 are logically incompatible but have a core in common. In that case, D1 and D2 can be mapped into E which corresponds to this common core provided that the community of interest does not care about the properties where they differ. Again, pragmata and special encoding conventions can handle the exceptional cases.

Note that case (3) is NOT necessarily the choice of the FRIML language community. The FRIML folk may decide that their language does not support inward mappings from either D1 or D2. Or they might decide that FRIML comes reasonably (and usefully) close to supporting D2 but NOT D1. Support for D1 AND D2 would arise in a community that finds both D1 and D2 desirable or necessary to support. Example: A set of procedures is implemented in Romberg, a language that comfortably supports internal analogues of D1 and D2. Application vendors that are porting to FRIML want to support the same procedure set without tinkering with the existing advertisements for the set.

You're probably quite right when you say that it is "unlikely that either the CLIPC or RPC will want to distinguish Range of Integer from Integer." You're certainly right if you're referring to the exchange encoding, i.e., it is not likely that anyone will want to attach a Range tag and the defined range to every value of Range of Integer that is actually exchanged. The marshalling routine or the unmarshalling routine COULD do a range check. But, it is more likely that the range checking will be left either to the type checking of L2 or to the code of the procedure itself.

On the Enumerated to Integer 4 to 7 issue, two points seem worth stressing:

(1) Strictly speaking, there is a loss of information in going from Enumerated to Range of Integer. We ignore this loss because we make the common assumption that the programmer, e.g., the writer of procedure P, KNOWS (has internalized and put implicitly into his code) what each of the symbolic values of Enumerated MEANS. For example, if the values are MONDAY, TUESDAY, WEDNESDAY, etc, he knows that MONDAY is the first day of the normal work week if that knowledge is relevant to the logic of his program. However, it is not hard to imagine a situation where the programmer doesn't know that the first value of the Enumeration type is MONDAY until he compares the value against a database where names of days and their characteristics are stored. Getting an Integer value of "one" wouldn't be much help if "one" could be MONDAY or SUNDAY, the choice were dynamic, and the days of the week were stored by name in the database! (As they might well be: Monday is "one" by some criteria, Sunday is "one" by others, etc).

(2) In any case, the "standard" that "one" corresponds to the first value of an Enumeration type is NOT part of CLIDT. It might be specified by CLIPCM, RPC, or some other community. From the CLIDT perspective, Enumerated (A, B, C, D) => Range of Integer (4-7) is just as good (or bad) as Enumerated (A, B, C, D) => Range of Integer (1-4).

Finally, the important point that is unresolved in my mind is where CLIDT formalism leaves off and other, e.g., CLIPCM IDN formalism, begins. If one carried the matter to its logical extreme, the formalism ought to support a full functional programming language or its declarative equivalent. After all, what the caller of a procedure really wants to know is not just the operands of the procedure but what the procedure DOES! Similarly, the caller of a server for an object-oriented database wants to know what the methods of a given class DO. But, I don't think anybody is advocating that even CLIPCM, let alone CLIDT, should go that far. On the other hand, when Mark Hamilton said, in Monterey, that CLIDT should be extended because there were types whose value spaces could not be defined by the existing formalism, he was coming perilously close to going exactly that far. The only obvious guideline I can think of is that CLIDT formalism should not favor CLIDT/RPC users at the expense of others (which seems to be what Hamilton has in mind when he talks about "direct compliance" of CLIDT syntax and CLIPCM syntax). On the other hand, there is no reason to avoid direct compliance just to be difficult!

Best Wishes,
Ed Greengrass

3/29/91

To: Ed Barkmeyer
From: Ed Greengrass
Subject: Mappings

Dear Ed

In our recent exchange regarding mappings, marshalling, Craig's view, etc., it occurs to me that I haven't stated my conclusion explicitly. Also, I've had a few additional thoughts. So, at the risk of inundating you, here goes.

Conclusion

I conclude that the text that I offered for a CLIDT section on mappings in clidt.90.293 is essentially correct except that additional statements about (1) inverses, and (2) who defines mappings, might be added. I think that the Craig view, as I understand it, is wrong for CLIDT, and probably wrong for CLIPCM, too.

Now, my additional comments:

You say that CLIDT conformance does not require unique inverse mappings, meaning that two CLIDT types D1 and D2 may be mapped inwardly into the same internal type E in language FRIML. However, in such a case, the outward mappings E => D1 and E => D2 are (or at any rate, can and should be) well-defined. The only thing that might not be well-defined is which of them to choose in a particular case. But, in most cases, including CLIPCM, it IS well defined. If procedure SNARK is defined to have an input parameter of CLIDT type D1, then (as you pointed out) FRIML procedure calls to SNARK will require an outward mapping from E to D1. Similarly, if SNARK has an output parameter of CLIDT type D2, then the outward mapping from a FRIML implementation of SNARK is from E to D2. There may possibly be a loss of meaningful information in performing these mappings but the mappings themselves are well-defined. Moreover, the mappings are not "local" if they were defined by the FRIML language community. They are not even local if they are defined by the CLIPCM/FRIML community though I don't see why they should be. They would be local ONLY if they were defined by the SNARK community. The SNARK folk would enter the picture if the mappings between D1, D2 and E were not generally useful but were acceptable in the SNARK context because of the peculiarities of the SNARK application.

The above considerations are by no means unique to CLIPCM or RPC. They would seem to apply to any case where the datatypes expected by the receiver are known to the sender, i.e., most protocols. There may be cases where the sender supplies unsolicited data that does not conform to any pre-agreed protocol but just carries its own definition with it. In such a case, a FRIML sender that wants to send E-type data has a choice of either outward mapping, E => D1 or E => D2. However, even in that case, there must be some kind of agreement regarding the schema that describes the data: if that agreement, includes D1 but not D2, the uncertainty is once again removed.

From: edbark@cme.nist.gov (Ed Barkmeyer)
To: Ed Greengrass
Subject: Re: Mappings

I don't think we disagree. When I said "local" I meant "local to the FRIML language", or possibly to a particular call, not to the user or station. And when you refer to the "outward" mapping from E to D1 and from E to D2, you are referring to what Craig calls the "inverse inward mapping", because it does not depend on E, it depends on D1 or D2. (An outward mapping would be determined by the internal datatype ONLY.)

Brian Meek would disagree that mapping E to both D1 and D2 on the outward side makes any sense at all. In his view, the outward mapping identifies the FRIML type E with the most appropriate CLI datatype having the semantics of E. If E maps to D1, then it cannot map to D2. Or as I understand it, E "means the same as" D1, although E "is an adequate representation of D2". (The latter is a statement of an "inward mapping".)

-Ed

4/1/91

To: Ed Barkmeyer
From: Ed Greengrass
Subject: Mappings Yet Again, (and Other Stuff)

Dear Ed.

Mappings

NO, NO, NO. I believe that a basic point of disagreement remains regarding mappings. I think your last message put the finger on that point. You said,

> when you refer to the "outward" mapping from E to
> D1 and from E to D2, you are referring to what
> Craig calls the "inverse inward mapping", because
> it does not depend on E, it depends on D1 or D2.

On the contrary, I believe that the outward mappings to which I am referring most emphatically DO depend on the internal types of language FRIML. The FRIML community should define an outward mapping to the CLI datatype, D, that best corresponds (hopefully, exactly corresponds) to E. If no datatype D corresponds exactly, then the FRIML folk MAY choose to define mappings to a datatype D1 that best approximates E or perhaps to two datatypes D1 and D2 that come about equally close. However, in such a case, a pragma would be required to define what is missing in the mapping, e.g., what E lacks that D (or D1 and D2) possess. For example a mapping from Integer to either Enumerated or State lacks names. Hence, the CLI version would have to have dummy names rather than user-defined names.

Some other community, e.g., the CLIPCM community, may choose to define mappings from languages L1, L2, etc, to CLI types commonly in use by advertised procedures. FRIML may be one of those languages. In that case, one might say that the mappings depend on (that is, are driven by) the CLI types D1, D2, etc, in the sense that the effort is to find the best FRIML approximations to D1, D2, etc, rather than the best CLI approximations to FRIML types E1, E2, etc. However, if using CLI types as a medium of exchange is to be meaningful, then it is still necessary to supply pragmata to define the differences between internal types and the CLI types to which they are outwardly mapped. It seems plain that the mapping depends on the internal types as well as on the CLI types. In fact, as I said earlier, the FRIML outward mappings for CLIPCM would probably be the product of a joint FRIML/CLIPCM effort.

In sum, I believe that I basically agree with Brian except that I recognize that mappings (outward as well as inward) may be defined by different communities with different needs and that approximations may be required and acceptable for some of those needs, AS LONG AS PRAGMATA DEFINE THOSE APPROXIMATIONS.

I'm going to try to send you an updated version of what I believe CLIDT should say about Mappings. If there is really no disagreement, then you and Craig should both be happy with my words.

From: edbark@cme.nist.gov (Ed Barkmeyer)
To: Ed Greengrass
Subject: Re: Mapping Yet Again (and Other Stuff)
Cc: edbark@cme.nist.gov, schaffert@crl.dec.com.,
udaa000%hazel.cc.kcl.ac.uk@nsfnet-relay.ac.uk

1. Inward mappings and outward mappings. I have the following notions:

a. There is only one kind of "inward mapping". If M is an inward mapping from CLID into FRIML, and D is a CLI datatype, then either M(D) is a datatype in FRIML, or M(D) is not defined. If M(D) is not defined, we say "FRIML cannot support CLI datatype D". If M: D -> E, then we say "E is the analog/support/ representation for CLI datatype D in FRIML". As you say, it may be advisable to "annotate" the mapping to say whether E is a real support, or a convenient but semantically inaccurate representation, and whether there are limitations. An inward mapping is a function, i.e. M(D) is unique.

b. There are two kinds of "outward mapping". If M is a "semantic outward mapping" from FRIML to CLID, and E is a datatype in FRIML, then M(E) is a CLI datatype (possibly a defined-datatype). If M: E ->

D, then we say "the semantics of E is D". If M(E) does not have a reasonable image in CLID, then there is a defect in the CLID. (Although it is possible that something which FRIML considers a "datatype" goes beyond the scope of "datatype" as defined in the CLID.) A semantic outward mapping is a function, i.e. M(E) is unique. A complete definition of a CLID defined-datatype provides for (requires) semantic "annotation"; and the "pragmata" provide for representation notions which are inherent in the FRIML datatype E.

c. If M is an "inverse inward mapping" (the second type of outward mapping) from FRIML to CLID, then there is an inward mapping M' from CLID to FRIML. If D is a CLI datatype and M': D -> E, then M: E -> D. In general, an inverse inward mapping is not a function, i.e. it is possible that M: E->D1 and M: E->D2. And if v is a value of type E in FRIML, M(v) is NOT well-defined. However, if a CLIPC interface defined by an IDN "schema" requires the second parameter of a particular routine to have CLI datatype D1, and the actual value is v (of FRIML type E), THEN M/D1: v -> v1 (of type D1). But this is precisely inverting M'(D1) at value v. And the "semantics" of M is precisely that it is the inverse of M'.

d. The FRIML standard, or some related standard, should define the "semantic outward mapping" of FRIML. Whether it should also define the "inward mapping" from CLID into FRIML is somewhat open. One would expect it to, but the CLIPC/ RPC community which uses FRIML would want to be heard in that regard. I don't think it is appropriate for the CLIPC/RPC group to define the inward mapping at all, but your suggestion of a joint effort seems best.

e. The CLIPC/RPC implementaiton will never use the semantic outward mapping. It will use the inward mapping and the inverse inward mapping.

f. The creators of service routines written in FRIML should make judicious use of the inward and the semantic outward mappings in defining their routines both in FRIML and in the IDN.

Can we agree on this? Or do we still disagree?

4/2/91

To: Ed Barkmeyer
From: Ed Greengrass
Subject: Mappings, the Final (?) Chapter

Dear Ed,

Yes, we're very close to agreement. One point: It seems to me that all mappings are as "semantic" as their specifiers can make them. In the best of all possible worlds (which alas we do not inhabit), the distinction between the two kinds of outward mappings would not exist. In the real world, it appears that non-unique mappings may have to be defined by service communities both outwardly and inwardly.

Case 1: CLIDT being more powerful than FRIML, M'(D1) => E and M'(D2) => E. This inward mapping M is still unique. The FRIML community can define a unique semantic outward mapping, e.g., M(E) => D2, the choice being arbitrary if there is no better reason. Service communities that want to use both D1 and D2 in their IDN's must define non-unique outward mappings which you call "inverse inward mappings".

Case 2: M'(D1) => E1 and M'(D2) => E2. However, the service community chooses, wisely or not, to support D1 but NOT D2. Hence, service calls from FRIML programs must define M(E1) => D1 and M(E2) => D1. This is a unique outward mapping but different from the semantic outward mapping. (More precisely, it is a weaker semantic mapping, the best that can be done with the given

constraints.) FRIML service programs will not use D2 as an operand in their IDN prototypes. They may choose not to use E2 in their FRIML prototypes either to remove ambiguity.

An example of case 2 would be a decision by CLIPCM not to support the CLI type Enumerated, and to map bot Enumerated and Integer into Integer.

A final quibble: The term "inverse inward mapping" is asking for trouble since the natural tendency (at least my natural tendency!) is to think of it as an inward mapping. Plainly, the term is a contraction of "inverse OF inward mapping".

Best Wishes,
Ed G.

From: edbark@cme.nist.gov (Ed Barkmeyer)
Message-Id: <9104031646.AA01340@radio.cme.nist.gov>
To: magnet@cs.UMD.EDU
Subject: Re: Mappings, the Final Chapter

I agree. I had not thought about the Case 2 problem, and I agree with your position. Your quibble is correct. I was trying to make clear what Craig must have meant by saying "you don't need outward mappings at all". He meant "for CLIPC/RPC you don't need to develop outward mappings separately - you can derive them from the 'inward' mappings". For my part, I don't care what it is called, or as Mozart said in "Amadeus", it can be in Turkish for all I care, as long as the music is there!

-Ed