From: Frank Farance
Organization: Farance Inc.
Telephone: +1 212 486 4700
Fax: +1 212 759 1605
E-mail: frank@farance.com
Date: 1995-12-22
Document Number: WG14/N530 X3J11/95-131
Subject: Issues on C Binding for LIPC

In the previous WG14/N463 document, ''Impact of adding WG11's LIA-1, LID, and LIPC features'', I had described adding LIPC (Language Independent Procedure Calling) as possible for C9X, assuming the LID (Language Independent Datatypes) issues were resolved. Since them, WG14 has had presentations from Craig Schaffert (of WG11) and Tom MacDonald (concerning Fortran compatibility).

The main problem is that there are two different notions of compatibility: (1) at the functional level -- arguments, calls, and results are described abstractly and mapped transparently into the target environments, (2) at the byte level -- bit/byte ordering and alignment, stack ordering and alignment, calling conventions, addressing conventsions, pointers, pass by value, pass by reference, pass by copy-in-out, pass on demand.

HEAVYWEIGHT PARADIGM

The first paradigm is a ''heavyweight'' paradigm that involves argument conversion, marshalling and unmarshalling parameter information. This paradigm is used for RPCs (remote procedure calls).

The heavyweight paradigm is unlikely to be solved in C9X. One indicator of this is WG14's lack of interest in solving bit/byte ordering and alignment issues for data structures -- necessary for communication among different architectures in networking, database, and distributed computing applications. If WG14 isn't interested in interoperability among C programs in different machine architectures, it certainly won't be interested in solving the problem with different languages across different machine architectures.

LIGHTWEIGHT PARADIGM

The second paradigm is a ''lightweight'' paradigm that is similar to the interoperability among programming languages (e.g., C, C++, Fortran, Pascal) within a single machine architecture. This paradigm involves either the caller or callee emulating the other's calling conventions (e.g., stack reordering and realignment).

We should solve the problems of the lightweight paradigm first: if we can't address the problem in a single machine architecture, then we certainly can't solve it among several machine architectures. There are several mechanisms

available (e.g., the use of keywords, such as "fortran", or C++'s linkage technique of using "extern "C" func()"). Since WG11 is solving an SC22 problem, we only need to solve the ''handshake'' problem for the number of languages within SC22. We should recognize that all solutions to the problem should involve the ''lightweight'' techniques described above.

RECOMMENDATION

WG14 should develop specific compatibility ''bridges'' between C and the common programming languages it interacts with (e.g., C++, Fortran, Pascal). There may be some interest in other programming languages (e.g., Cobol and LISP), but we should priorize the languages of interest. Certainly, our solution to the smaller problem must be contained within the abstraction of the larger problem (i.e., the heavyweight paradigm of LIPC), so we aren't losing anything (or future LIPC compliance) by solving the smaller problem first.