Programmable Applications in a Heterogeneous and Concurrent Environment

Paul F. Dubois

This paper was prepared for submittal to the
Tools U.S.A. 1995
Santa Barbara, CA
August 2, 1995

July 21, 1995

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.
DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Programmable Applications in a Heterogeneous and Concurrent Environment

Tools U.S.A. 1995
Paul F. Dubois

Inertial Confinement Fusion Program
Lawrence Livermore National Laboratory
L-472
Livermore, CA 94550 U.S. A.
dubois1@llnl.gov
Consulting: dubois1@ix.netcom.com

This work was produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL.
Equipe Basis is a new system for programmable applications which is under development at Lawrence Livermore National Laboratory.

Our system “Basis” for producing applications which can be programmed by the user needs modernization.

We view Eiffel as a key technology to make this possible.

This report is strictly a “work in progress” report and the final product may differ substantially. It is hoped that the presentation might stimulate conversation and helpful suggestions.
We have begun development of a next-generation system for programmable applications.

- Equipe Basis (EB) is designed to permit user control of teams of intercommunicating processes in a heterogeneous environment.
- An application is “programmable” if the user can control the behavior of the application with a full-scale programming language. Experience has proven that this concept greatly improves the productivity and power of both code groups and their users.
- Current systems work with programs written in Fortran or C on a single processor. The programs of the future will be in many languages and distributed over many processors.
The solution is to use object-oriented technology to create a user-extensible system with a **lingua franca** for communication.

- The object-oriented kernel can communicate data and commands between processes that are unaware of each other’s inner structure by using a kind of “InterGalactica”.

- The system is easily extended to support new user types or languages because the kernel does its work via polymorphic dispatch.

- Fast message passing is still available to processes that “know” each other. Think of a telephone net (the message passing) vs. an interstate highway system (the programmable control). Both are needed.

```
Message passing is intimate.
```

```
InterGalactica Here
```

```
Pkg 1
```

```
Pkg 2
```

```
Graphics
```

Programmable applications have greatly increased productivity.

The user controls the program with a programming language. We wrote an array-Fortran-like language that would be familiar to scientists which won quick acceptance.

The key variables and procedures in the compiled program are “built-in” to this language so the user can do all input, output, and control with the user interface.

Over ten years we have proven that this:

- Enhances user productivity and empowers their creative use of the application.
- Greatly shortens development time for new physics.
- Greatly reduces requests for modifications and additions to the program.
- Allows automation of user tasks, including formulating the next job in terms of the results of the current one.
Our system, Basis, was possible because it was omniscient and omnipotent.

Begun in 1984, it faced only a world of Fortran.

It “knew” all possible data types, their storage requirements, what operations were defined on them, etc.

It “knew” how to pass arguments to routines and get results back.

It “knew” there was only one CPU and one address space.

Obviously, this world is disappearing. But the concept of programmable applications has proven valuable. How do we translate this into the new world of heterogeneous platforms and parallel processing?
Object-oriented technology is the key to programmable applications in heterogeneous environments.

If I don't know what language you are written in, or what kind of machine you are running on, all I can do is send you messages about what I want done.

These messages must be in some core system of commonly understood classes.

A C++ class can take data from a Fortran module if the Fortran module translates it into a common language and the C++ class gets it out.

We envision a communication agent and a database manager based on this minimal kernel as a part of each process.
We have begun writing the kernel of this system by constructing a set of Eiffel classes which will represent the lingua franca.

We want a big enough set, with enough class structure, to make expression easy and polymorphism/dynamic binding really useful.

We can use EiffelNet and the STORABLE classes as immediate means to get the prototype up and doing something useful.

We have chosen Eiffel for many reasons, but the fact that it is a small, easy language for physicists is one of them.

Garbage collection and exception handling will make writing the language parser much easier than the one we wrote for Basis.
Our basic classes have evolved to look almost like a replacement for the basic classes of Eiffel.

Some constraints on the solution
You don’t really want to fool with expanded classes for this kind of environment.

You want to express the fact that scalars and arrays are the primary items of interest for scientists.

On the other hand, you can’t possibly guess what kind of data structures are going to be needed.

You want to make principal operations in a control language easy to express via polymorphism.
The InterGalactica inheritance hierarchy

BASIC_EB
  COMPOSITE_EB
    ARRAY_EB [T]
      ARRAY_STRING_EB
      ARRAY_REAL_EB
      ARRAY_INTEGER_EB
      ARRAY_DOUBLE_EB
      ARRAY_CHARACTER_EB
      ARRAY_BOOLEAN_EB
    STRING_EB
    RANGE_EB
    LIST_EB [T -> BASIC_EB]
  SCALAR_EB
    SCALAR_NUMERIC_EB
      FLOAT_EB
        REAL_EB
        DOUBLE_EB
        INTEGER_EB
    POINTER_EB
    NULL_EB
    CHARACTER_EB
    BOOLEAN_EB
  NUMERIC_EB
    SCALAR_NUMERIC_EB...
    ARRAY_NUMERIC_EB
      ARRAY_REAL_EB
      ARRAY_INTEGER_EB
      ARRAY_DOUBLE_EB
The top of the cluster is BASIC_EB.

indexing
description:
"Those classes that can be used as standard base classes"

defered class BASIC_EB inherit

    EQUIPE_BASIS
    undefine
    out
    end

feature -- Conversion

    meta_out: STRING is
        -- Describe the object.
        do
            ....
        end;
Serial access is provided for all types.

item_eb (i: INTEGER): BASIC_EB is
    -- The i_th item, based on 1
    require
        valid_index: valid_index_eb (i)
        deferred
    end

valid_index_eb (i: INTEGER): BOOLEAN is
    -- Is `i` suitable for item_eb?
    do
        Result := i >= 1 and i <= count
    end;

count: INTEGER is
    -- Number of items in the composite
    deferred
    end;

eb_item (s: BASIC_EB; i: INTEGER) is
    -- Assign a new value to the i_th component.
    require
        valid_index: valid_index_eb (i);
        deferred
    end;
eb_item_failed: BOOLEAN;
    -- Did the last eb_item succeed?
Shape and subscripting functions can be done once at this level, using item_{eb} for access.

feature -- Shape

    shape: SHAPE_{EB} is
        -- Current effective shape.
        do
            !! Result.make (1, count);
        end;

    subscript_count: INTEGER is
        -- Number of elements in shape.
        do
            Result := shape.count
        ensure
            correct: Result = shape.count
        end;

    subscript_lower (i: INTEGER): INTEGER is
        -- Lowest valid subscript in i’th dimension
        require
            valid_subscript: valid_subscript (i)
        do
            Result := shape.item (i).lower
        end;
Each item of interest will be represented by a wrapper.

A wrapper is a named "agent" which responds to a standard set of messages.

A wrapper for a Fortran array, for example, can report its size, and return individual elements or entire arrays as a BASIC_EB quantity.

Polymorphism allows the disparate wrappers to translate to and from InterGalactica each in their own way.
Here is a small example.

```plaintext
local
d: DB_EB[WRAPPER_EB];
nw, nx, ny: DOUBLE.WRAPPER_EB;
do
!! d.make("global");

!! nw.make("x", 3.)
d.put(nw);
!! nw.make("y", 5.)
d.put(nw);

nx ?= d.item("x");
yx ?= d.item("y");
if nx = Void or ny = Void then
  raise("Error in getting nx or ny")
end;

nx.from_basic(
  double_eb(
    nx.to_basic.item +
    ny.to_basic.item)
)
```

"to_basic" yields a DOUBLE_EB, which is a child of DOUBLE_REF.

The assignment attempt allows us to discriminate between different kinds of BASIC_EBs.
The user interface will be a language interpreter written in Eiffel.

In the present system this interpreter is part of the program in which the physics lives. We envision that in the future, it will be a separate process communicating via InterGalactica.

The language will be similar to Basis, Fortran-array like with structures.

Only a small database portion need be resident in the physics code. This is equivalent to R2D2’s “data port”.

The parser itself is created using an LR(1) parser generator that produces an Eiffel parser for the language, ZLR. (Zane’s LR, written by Zane Motteler).

This parser can translate InterGalactica into input formats needed by commercial tools such as AVS. It becomes a “universal translator”.