Toward a Verifiable Approach to the Design of Concurrent Computations

The High-Performance Computing Symposium – 1993

G. H. Chisholm
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439

ABSTRACT

Distributed programs are dependent on explicit message passing between disjoint components of the computation. This paper is concerned with investigating an approach for proving correctness of distributed programs under an assumed data-exchange capability. Stated informally, the data exchange assumption is that every message is passed correctly, i.e., neither lost nor corrupted. One approach for constructing a proof under this assumption would be to embed an abstract model of the data communications mechanism into the program specification. The Message Passing Interface (MPI) standard provides a basis for such a model. In support of our investigations, we have developed a high-level specification using the ASLAN specification language. Our specification is based on a generalized communications model from which the MPI model may be derived. We describe the specification of this model and an approach to the specification of distributed programs with explicit message passing based on a verifiable data exchange model.

1. INTRODUCTION

Distributed programs consist of a collection of concurrent processes that communicate by explicit message passing. Such programs have been observed to fail on loss or corruption of data during the point-to-point communication of messages. This situation suggests that verification of distributed programs should include a proof that the program satisfies a data exchange property. The following describes progress in developing an approach for the specification and verification of distributed programs. A proof that a distributed program is correct with respect to assumed data-exchange properties is beyond the scope of this paper. However, we describe an essential element for the development of distributed programs that satisfy this property.

The literature includes a number of approaches to program verification. Our paradigm is based on an axiomatic reasoning approach as described in [4, 7, 5]. This approach is predicated on an assertion language and proof system. The assertion language allows

representation of the program and construction of formal proofs of relevant properties all within the proof system. In some applications, the proof rules are constructed to support verification of existing programs. In other applications, the proof rules are constructed to support specification of properties and verification that code satisfies its specification.

One property of interest to those verifying concurrent programs is "fairness." The fairness assumption states that every component of a concurrent program is eventually activated. Reference [1] describes an axiomatic method for verification of the fairness assumption. This method is predicated on the embedding of a schedular into a nondeterministic program and constructing the proof.

We suggest an extension to this approach for dealing with the data-exchange assumption. The data exchange assumption is that every message is passed correctly–that is, no message is lost or corrupted. Specifically, we suggest using a specification-based proof system. The approach is predicated on embedding a specification for an abstract data exchange mechanism into the program and constructing a proof of correctness based on this specification. In this paper we describe the specification of such a mechanism. This specification was written in the ASLAN specification language [2].

Note that we describe a high-level specification: the level of detail is insufficient to support a proof of the data exchange assumption. However, it represents the initial effort in achieving our ultimate goal of supporting such a proof. The next phase is to develop a multilevel ASLAN specification with successively more detail. One constraint on this development should be to assure that the specification is applicable to a diverse set of applications, in particular, those of interest to the high-performance computing community.

We also recommend that the ASLAN specification presented here be further developed to include the a Message-Passing Interface (MPI) standard currently under development [3, 6]. The development of a second formal description of the MPI model would provide additional confirmation of the correctness and completeness of the model and would increase the accessibility of that model.

2. DESCRIPTION OF A GENERAL COMMUNICATIONS MODEL

In this section we present a description of a high-level specification of a general message-passing model based on the ASLAN specification language [2]. We begin with a brief description of ASLAN and then discuss code for an ASLAN specification of a communications model. The complete specification is contained in the Appendix.

2.1 ASLAN Specification Language

ASLAN is based on the first-order predicate calculus with equality. The system being specified is considered to be in various states, depending on the values of the state variables. Changes in state variables take place only by well-defined transitions. ASLAN uses induction to show (1) that the system defined by these state variables and transitions always satisfies some critical requirements, and (2) the resulting state after a transition also satisfies the invariant assertion.

2.2 INITIALIZATION

The following is analogous to the declaration section of a program and supports the strong typing of the ASLAN language.

```
TYPE
 DataGram,
 DataGrams IS SET OF DataGram,
 Message,
 Location,
 Locations IS SET OF Location,
 Precedence_Level IS TYPEDEF T:Integer(T >= 1 & T <=3),
 Message_Type,
 Location_Type,
 Time
```

We use the term "DataGram" to refer to anything that floats around in the ether. The entire bandwidth of the communications network is finite and may be envisioned as being made up of either DataGrams or not-DataGrams. Here we focus only on failures to communicate DataGrams; properties of not-DataG rams are outside of the current model. One mode of failure would be to consider something to be a DataGram when it is not. A second failure would be to have a valid DataGram and to fail to communicate it. The bandwidth of all possible messages is countable finite and modeled as a set made up of "DataGrams" and "-DataGrams." These are properties of the system that are outside of the current model. The intent of presenting this discussion is to demonstrate the necessity of carefully selecting the data representation in anticipation of capturing diverse system properties in a specification.

```
CONSTANT
 TO(DataGram): Location,
 FROM(DataGram): Location,
 Data(DataGram): Message,
 Timestamp(DataGram): Time,
 Precedence(Message): Precedence_Level,
 Type_of_Message(Message): Message_Type,
 Consistent_Message(Message_Type,Location_Type): Boolean,
 Type_of_Site(Location): Location_Type,
 Well_Formed(DataGram): Boolean,
 Neighbor(Location,Location): Boolean


VARIABLE
 Send_Buffer(Location):DataGrams,
 Rec_Buffer(Location):DataGrams,
 Network:Datagrams,
 Action_Items(Location):DataGrams,
 Now : Time

INITIAL
   FORALL L:Location (
      Send_Buffer(L) = Empty
     & Rec_Buffer(L) = Empty )
   & Network = Empty
```

2.3 INVARIANT

Recall that an invariant defines requirements that must be met in every reachable state. Intuitively, the critical requirement is that a DataGram can be in one place at a time.

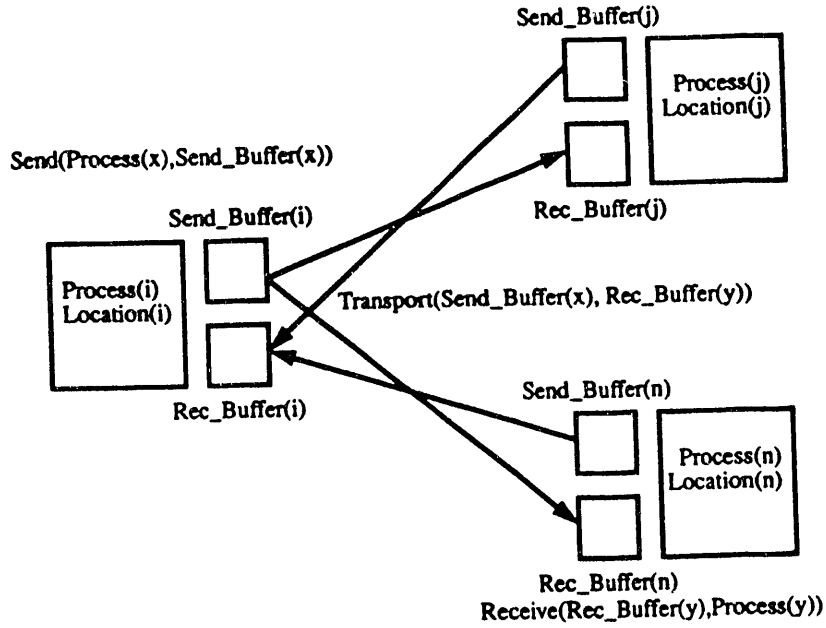
```
FORALL D:Datagram (
    (EXISTS L1:Location(D ISIN Send_Buffer(L1))
       -> FORALL L2:Location (
            D ~ISIN Rec_Buffer(L2) & D ~ISIN Network) )
  & (EXISTS L1:Location(D ISIN Rec_Buffer(L1))
       -> FORALL L2:Location (
            D ~ISIN Send_Buffer(L2) & D ~ISIN Network) )
  & (D ISIN Network
       -> FORALL L:Location (
            D ~ISIN Rec_Buffer(L) & D ~ISIN Send_Buffer(L)))
```

```
& FORALL L1,L2:Location (
          D ISIN Rec_Buffer(L1) & D ISIN Rec_Buffer(L2)
               -> L1=L2 )
& FORALL L1,L2:Location (
          D ISIN Send_Buffer(L1) & D ISIN Out_Buffer(L2)
               -> L1=L2 )   )
```

2.4 CONSTRAINTS

Figure 1 is a block diagram of a general communications model. Depicted in Figure 1 is the interface between processes and the data-exchange mechanism. A process passes data to the Send_Buffer for eventual incorporation into a DataGram by the "Send" function. The "Transport" function moves DataGrams from the Send_Buffer of one location to the Rec_Buffer at another location. The "Receive" function removes a DataGram from a Rec_Buffer for presentation to a process at that location.

Figure 1 Block Diagram of an Abstract Data-Exchange Mechanism



```
FORALL L:Location, D: DataGram (
   (D ISIN Rec_Buffer(L) & D ~ISIN Rec_Buffer'(L) ->
          ( EXISTS L1:location (D ISIN Send_Buffer'(L1))
           | D ISIN Network')
        & FORALL D1:DataGram (
           (  EXISTS L1:Location(D1 ISIN Send_Buffer'(L1))
            | D1 ISIN Network')
         -> Precedence(Data(D)) >= Precedence(Data(D1)) ))
& ...
```

This states that for all locations, L, IF a DataGram is in the input buffer or network before a transition fires, but is not in the same buffer after firing, THEN the precedence for that DataGram is equal to or higher than all DataGrams eligible for processing by "Transport."

In such a system, "Transport" and "Receive" are autonomous. Selection of which DataGram is to be processed is dependent on precedence. All DataGrams processed by "Receive" were of equal or higher precedence than those available to "Transport."

In contrast, the "Send" function is conceived as sequentially processing messages from an application. This constraint extends the generic model to encompass a distributed system with multiple precedences associated with messages, that is, systems that may simultaneously be processing real, simulated, or test data.

```
  (D ISIN Send_Buffer'(L) & D ~ISIN Out_Buffer(L) ->
     ( EXISTS L1:location (D ISIN Rec_Buffer(L1))
        | D ISIN Network)
   & FORALL D1:DataGram (
     (  EXISTS L1:Location(D1 ISIN Send_Buffer'(L1))
         | D1 ISIN Network')
        -> Precedence(Data(D)) >= Precedence(Data(D1)) ))
& ...
```

All DataGrams processed by "Transport" were of equal or higher precedence that those eligible for "Transport."

```
(D ISIN Rec_Buffer'(L) & D ~ISIN In_Buffer(L) ->
     FORALL D1:DataGram ( D1 ISIN Rec_Buffer'(L)
    -> Precedence(Data(D)) >= Precedence(Data(D1))))
& ...
```

```
D ISIN Action_Items(L) & D ~ISIN Action_Items'(L)
        -> Well_Formed(D)
```

Only well formed DataGrams are "Received."

The constraint is the conjunction of these three constituents:

1. all DataGrams processed by "Receive" and were of equal or higher precedence than those available to "Transport'," and
2. all DataGrams processed by "Transport" were of equal or higher precedence that those eligible for "Transport," and
3. only well formed DataGrams are "Received."

2.5 THE SEND FUNCTION

```
TRANSITION Send ( Sender:Location, Receivers:Locations,
          M:Message )
 ENTRY
  Consistent_Message(Type_of_Message(M),
                     Type_of_Site(Sender))
 EXIT

  Send_Buffer(Sender) BECOMES Send_Buffer'(Sender) UNION
   {SETDEF D:DataGram
      EXISTS R:Location ( R ISIN Receivers
    & TO(D) = R
    & FROM(D) = Sender
    & Data(D) = M
    & Timestamp(D) = Now ) }
```

Input to the transition consists of "Sender," a list of "Receivers," and a "Message." An entry conditions states that the message must be consistent for the site. Firing the transition results in the generation of a new DataGram that is appended to the "Send_Buffer" for a Location/Sender. A DataGram is formed for each Receiver and is made up of "TO,", "FROM," "Data," and "Timestamp."

2.6 THE TRANSPORT FUNCTION

```
TRANSITION Transport ( D:DataGram )
ENTRY
  EXISTS L:Location (D ISIN Send_Buffer(L)) | D ISIN Network
```

```
& FORALL D1:DataGram (
    EXISTS L1:Location (  D1 ISIN Send_Buffer(L1))
                        | D1 ISIN Network
                             -> Precedence(Data(D))
                                >= Precedence(Data(D1)))
EXIT
    IF EXISTS L:Location (D ISIN Send_Buffer'(L))
        THEN UNIQUE L:Location (
            D ISIN Send_Buffer'(L)
          & Send_Buffer(L) BECOMES
                           Out_Buffer'(L) SET_DIFF {D}
          & IF Neighbor(TO(D),L)
              THEN Rec_Buffer(TO(D)) BECOMES
                                     In_Buffer'(TO(D))
                                     UNION {D}
              ELSE Network = Network UNION {D}
            FI )
        ELSE (  Rec_Buffer(TO(D)) BECOMES
                                  In_Buffer'(TO(D))
                                  UNION {D}
              & Network = Network SET_DIFF {D}
              | NOCHANGE(Rec_Buffer,Network) )
    FI
```

The firing of the "Transport" transition results in a DataGram being moved from a Send_Buffer to a Rec_Buffer. The entry condition is that there is a DataGram available for processing. On exit, Send_Buffer' is the result of removing the DataGram, and Rec_Buffer'(TO) has the DataGram appended to it. All actions are consistent with the precedencehandling protocol.

2.7 THE RECEIVE FUNCTION

```
TRANSITION Receive (Receiver:Location, D:DataGram)
 ENTRY
     D ISIN Rec_Buffer(Receiver)
  & FORALL D1:DataGram (Precedence(Data(D)) >=
                        Precedence(Data(D1)))
 EXIT
    Rec_Buffer(Receiver) BECOMES Rec_Buffer'(Receiver)
```

```
                                    SET_DIFF {D}
& IF Well_Formed(D)
 THEN Action_Items(Receiver) BECOMES
   Action_Items'(Receiver) UNION {D}
 FI
```

Inputs are Receiver and DataGram. The DataGram is equal or higher in precedence than others in the Rec_Buffer. Following firing of the transition, DataGram has been removed from the Rec_Buffer, and, if Well_Formed, moved into the Action_Items buffer. This is envisioned as being passed off to an application.

3. CONCLUSIONS AND RECOMMENDATIONS

Distributed programs consist of a collection of concurrent processes that communicate by explicit message passing. Such programs have been observed to fail on loss or corruption of data during the point-to-point communication of messages. This suggests that verification of distributed programs should include a proof that the program satisfies a data exchange property.

We recommend that the ASLAN specification presented here be further developed to include the detailed MPI model. The benefit of accomplishing this work would be twofold. First, development of a second formal description of the MPI model would provide additional confirmation of the correctness and completeness of the model. Second, multiple representations of any model increases the accessibility of that model.

ACKNOWLEDGEMENTS

Thanks to Dick Kemmerer for assistance in development of the ASLAN specification and to Bret Michael and Gail Pieper for their diligent reviews and comment.

This work was supported by the National Security Agency, V31, under MOD 708992.

Bibliography

[1] Krzysztof R. Apt and Ernst-Rudiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer-Verlag, New York, 1991.

[2] Brent Auernheimer and Richard A. Kemmerer. ASLAN user's manual. Technical Report TRCS84-10, University of California - Santa Barbara, Santa Barbara, CA, April 1992.

[3] Jack J. Dongarra, Rolf Hempel, Anthony J. G. Hey, and David W. Walker. A proposal for a multi-level message-passing interface standard. Technical Report ORNL/TM-12231, Oak Ridge National Laboratory, Oak Ridge, Tenn., January 1993.

[4] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Society, Providence, R.I., 1967. Proc. Symp. Appl. Math. 19.

[5] D. Gries. *The Science of Programming*. Springer-Verlag, New York, 1981.

[6] William D. Gropp and Ewing L. Lusk. A test implementation of the MPI draft message-passing interface standard. Technical Report ANL-92-47, Argonne National Laboratory, Argonne, Ill., December 1992.

[7] Zohar Manna. *Mathematical Theory of Computation*. McGraw-Hill Inc., New York, 1974.

Appendix ASLAN SPECIFICATION OF COMMUNICATIONS MODEL

```
SPECIFICATION COMMUNICATIONS  /* 14 MAR 93 */
LEVEL Top_Level

TYPE
     DataGram,
     DataGrams IS SET OF DataGram,
     Message,
     Location,
     Locations IS SET OF Location,
     Precedence_Level IS TYPEDEF T:Integer(T >= 1 & T <=3),
     Message_Type,
     Location_Type,
     Time

     /* A Message is part of a DataGram - A DataGram includes information */
     /* such as sender, receiver and time stamp. */


CONSTANT
     TO(DataGram): Location,
     FROM(DataGram): Location,
     Data(DataGram): Message,
     Timestamp(DataGram): Time,
     Precedence(Message): Precedence_Level,
     Type_of_Message(Message): Message_Type,
     Consistent_Message(Message_Type,Location_Type): Boolean,
     Type_of_Site(Location): Location_Type,
     Well_Formed(DataGram): Boolean,
     Neighbor(Location,Location): BooleanVARIABLE
     Out_Buffer(Location):DataGrams,
     In_Buffer(Location):DataGrams,
     Network:Datagrams,
     Action_Items(Location):DataGrams,
     Now:Time

INITIAL
       FORALL L:Location (
             Out_Buffer(L) = Empty
           & In_Buffer(L) = Empty )
     & Network = Empty

INVARIANT
     FORALL D:Datagram (
          (EXISTS L1:Location(D ISIN Out_Buffer(L1))
                -> FORALL L2:Location (
                      D ~ISIN In_Buffer(L2) & D ~ISIN Network) )
        & (EXISTS L1:Location(D ISIN In_Buffer(L1))
                -> FORALL L2:Location (
                      D ~ISIN Out_Buffer(L2) & D ~ISIN Network) )
        & (D ISIN Network
                -> FORALL L:Location (
                      D ~ISIN In_Buffer(L) & D ~ISIN Out_Buffer(L)) )
        & FORALL L1,L2:Location (
                D ISIN In_Buffer(L1) & D ISIN In_Buffer(L2) -> L1=L2 )
        & FORALL L1,L2:Location (
                D ISIN Out_Buffer(L1) & D ISIN Out_Buffer(L2) -> L1=L2 )  )
```

```
CONSTRAINT
     FORALL L:Location, D: DataGram (
           (D ISIN In_Buffer(L) & D ~ISIN In_Buffer'(L) ->
                 ( EXISTS L1:location (D ISIN Out_Buffer'(L1))
                  | D ISIN Network')
                & FORALL D1:DataGram (
                        (  EXISTS L1:Location(D1 ISIN Out_Buffer'(L1))
                         | D1 ISIN Network')
                       -> Precedence(Data(D)) >= Precedence(Data(D1)) ))
      &
           (D ISIN Out_Buffer'(L) & D ~ISIN Out_Buffer(L) ->
                 ( EXISTS L1:location (D ISIN In_Buffer(L1))
                  | D ISIN Network)
                & FORALL D1:DataGram (
                        (  EXISTS L1:Location(D1 ISIN Out_Buffer'(L1))
                         | D1 ISIN Network')
                       -> Precedence(Data(D)) >= Precedence(Data(D1)) ))
      &
           (D ISIN In_Buffer'(L) & D ~ISIN In_Buffer(L) ->
                FORALL D1:DataGram ( D1 ISIN In_Buffer'(L)
                      -> Precedence(Data(D)) >= Precedence(Data(D1)) ) )
      &
           (D ISIN Action_Items(L) & D ~ISIN Action_Items'(L) ->
                Well_Formed(D) ))
RANSITION Send ( Sender:Location, Receivers:Locations, M:Message )
     ENTRY
           Consistent_Message(Type_of_Message(M), Type_of_Site(Sender))
     EXIT

           Out_Buffer(Sender) BECOMES Out_Buffer'(Sender) UNION
                {SETDEF D:DataGram
                    EXISTS R:Location ( R ISIN Receivers
                      & TO(D) = R
                      & FROM(D) = Sender
                      & Data(D) = M
                      & Timestamp(D) = Now ) }
TRANSITION Transport ( D:DataGram )
     ENTRY
          EXISTS L:Location (D ISIN Out_Buffer(L)) | D ISIN Network
        & FORALL D1:DataGram (
            EXISTS L1:Location (D1 ISIN Out_Buffer(L1)) | D1 ISIN Network
                              -> Precedence(Data(D)) >= Precedence(Data(D1)))
     EXIT
          IF EXISTS L:Location (D ISIN Out_Buffer'(L))
               THEN UNIQUE L:Location (
                   D ISIN Out_Buffer'(L)
                 & Out_Buffer(L) BECOMES Out_Buffer'(L) SET_DIFF {D}
                 & IF Neighbor(TO(D),L)
                     THEN In_Buffer(TO(D)) BECOMES In_Buffer'(TO(D)) UNION {D}
                     ELSE Network = Network UNION {D}
                   FI )
               ELSE (    In_Buffer(TO(D)) BECOMES In_Buffer'(TO(D)) UNION {D}
                       & Network = Network SET_DIFF {D}
                      | NOCHANGE(In_Buffer,Network) )
          FITRANSITION Receive (Receiver:Location, D:DataGram)
     ENTRY
          D ISIN In_Buffer(Receiver)
        & FORALL D1:DataGram (D1 ISIN In_Buffer(Receiver)
             -> (Precedence(Data(D)) >= Precedence(Data(D1))))
     EXIT
          In_Buffer(Receiver) BECOMES In_Buffer'(Receiver) SET_DIFF {D}
        & IF Well_Formed(D)
             THEN Action_Items(Receiver) BECOMES
```

```
                    Action_Items'(Receiver) UNION {D}
            FI
END Top_Level
```

END

DATE
FILMED
7/12/93