Report Number: WUCS-82-6

1982-02-01

A Formal Treatment of Distributed Systems Design

Authors: Gruia-Catalin Roman and Robert K. Israel

The paper reports on a technique for the formal definition of the distributed systems design methodology called the Total System Design (TSD) Methodology. Central to the formalization of the TSD Methodology is the TSD Model, which consists of several information structures and a set of consistency constraints (i.e., acceptance criteria). While the major part of this paper is taken by the model definition, the authors' intent is not simply to justify the model itself. The paper provides convincing evidence that rigorous methodology definitions are both feasible and useful. It offers examples of how to approach the formalization of various methodological concepts such as satisfiability of system requirements, hierarchical structuring of the system, and stepwise refinement. Finally, it shows how these and other concepts of practical value in distributed systems design may become better understood through the use of the proposed formalizations.
... **Read complete abstract on page 2.**

A Formal Treatment of Distributed Systems Design

Complete Abstract:

The paper reports on a technique for the formal definition of the distributed systems design methodology called the Total System Design (TSD) Methodology. Central to the formalization of the TSD Methodology is the TSD Model, which consists of several information structures and a set of consistency constraints (i.e., acceptance criteria). While the major part of this paper is taken by the model definition, the authors' intent is not simply to justify the model itself. The paper provides convincing evidence that rigorous methodology definitions are both feasible and useful. It offers examples of how to approach the formalization of various methodological concepts such as satisfiability of system requirements, hierarchical structuring of the system, and stepwise refinement. Finally, it shows how these and other concepts of practical value in distributed systems design may become better understood through the use of the proposed formalizations.

A FORMAL TREATMENT OF DISTRIBUTED SYSTEMS DESIGN

Gruia-Catalin Roman

Robert K. Israel

WUCS-82-6

February 1982

Department of Computer Science
Washington University
St. Louis, Missouri 63130

Presented at the Symposium on Current Issues of Requirements Engineering Environments, September 20 and 21, 1982, Kyoto, Japan.

As appeared in Requirements Engineering Environments, Y. Ohno (editor), OHM/North-Holland Pub. Co., 1982, pp. 3-12.

A FORMAL TREATMENT OF DISTRIBUTED SYSTEMS DESIGN

Gruia-Catalin Roman and Robert K. Israel

Department Of Computer Science
Washington University
Saint Louis, Missouri 63130

Abstract

The paper reports on a technique for the formal definition of a distributed systems design methodology called the Total System Design (TSD) Methodology. Central to the formalization of the TSD Methodology is the TSD Model, which consists of several information structures and a set of consistency constraints (i.e., acceptance criteria). While the major part of this paper is taken by the model definition, the authors' intent is not simply to justify the model itself. The paper provides convincing evidence that rigorous methodology definitions are both feasible and useful. It offers examples of how to approach the formalization of various methodological concepts such as satisfiability of system requirements, hierarchical structuring of the system, and stepwise refinement. Finally, it shows how these and other concepts of practical value in distributed systems design may become better understood through the use of the proposed formalizations.

Acknowledgements: This work was partially supported by Rome Air Development Center and by Defense Mapping Agency under contract F30602-80-C-0284. The contributions of W. E. Ball, W. D. Gillett and M. J. Stucki in discussing and reviewing parts of this paper are also acknowledged.

Introduction

The formalization of a system design methodology, like the semantic definition of a programming language, is central to the establishment of a correct usage and implementation of the methodology. As such, a formal treatment must involve the identification of the ordering and nature of design activities, the nature of the design specifications produced, and the acceptance criteria which these specifications must meet.

Our approach to modeling the system design process is similar to that of the operational models employed in the semantic definition of programming languages. The operational models consist of a set of information structures with an abstract interpreter which modifies them in accordance with a predetermined set of rules [7]. Similar information structures may be used to abstract the nature of the specifications generated by a design methodology. Design activities (synthetic and analytic) may be defined as transformations and predicate evaluations over the information structures with their sequencing being specified in some operational manner such as that proposed in [4]. Finally, the acceptance criteria applied to the delivered specifications may be defined by an additional set of predicates over the information structures.

This approach has been used successfully in the formal definition of the Total System Design (TSD) Methodology, which is a methodology for the design of distributed hardware/software systems [5]. Central to this definition is the TSD Model, which consists of several information structures and a set of consistency constraints (i.e. acceptance criteria) over those structures.

The TSD Model has its roots in three rather different technical areas: Alford's work on formal foundations for system specification and design [1], programming language semantics [7], and General Systems Theory [8]. The model developed by Alford shares a certain commonality of goals with the TSD Model (i.e. a better understanding of the system design process), and hence had a great influence on the development of the TSD Model. The two approaches differ, however, both in their perception of the system design process and in the formalisms employed. Alford's approach follows very closely the techniques seen in general systems theoretic models. Our strategy deviates from them in two important ways. First, the TSD Model focuses on concepts specific to distributed systems design and represents them directly as part of the model in ways that stress a clean separation of concerns. Second, the model borrows heavily from work in the area of programming language semantics. The similarity of the approach to that of operational semantics was described above.

Due to space limitations, this paper is concerned solely with the information structures and the consistency constraints that have been employed in the semantic definition of the TSD Methodology and the formalization of related design concepts. The TSD methodology is briefly outlined in the next section as a means of providing the reader with some of the motivation behind the development of the formal model.

The main body of the paper, however, is dedicated to a definition and discussion of the three information structures that are part of the model and the formalization of several important methodological concepts. These information structures are the system requirements, the processing model which captures the nature of the design specifications generated during system-level design (i.e., prior to the commitment to a particular hardware and software mix), and the hardware/software requirements which reflect the ultimate result of the hardware/software trade-offs taking place during system design.

Following the definition of the information structures, the consistency constraints which formalize several basic concepts of the TSD Methodology are presented. The nature of these constraints will be identified in the discussion of the information structures and the TSD Methodology below.

TSD Methodology Outline

The TSD Methodology is limited in scope to that part of the design which we call the system design stage. It covers all design activities involved in taking a set of overall system design requirements and generating the specification of the hardware and software requirements for the system. There are two phases that make up this stage: the system architecture design phase and the system binding phase. The former deals with the selection of a system architecture which accomplishes the intended system functionality and which, under a reasonable set of technological assumptions, meets the constraints (such as performance) imposed by the system requirements. The proposed architecture and all the design decisions taken during this phase form a processing model used as input to the binding phase.

The binding phase, based on the limited degrees of freedom still left open by the system architecture design phase and based on market availability, identifies a particular mix of software and hardware needed to implement the system and produces specifications for all needed components. The nature of the specifications, however, may vary from component to component depending on its intended implementation (software or hardware) and on the manner in which it is to be obtained (off-the-shelf, through customization, or custom-made). The system design stage is also concerned with the integration of the system components from the point when both the software and the hardware components are available and up to the point when the system is offered for customer acceptance testing.

The TSD methodology approaches the system architecture phase by treating the design of distributed systems in terms of the top down development of a hierarchy of design specifications that form a processing model. Each design specification corresponds to a subsystem of the overall system architecture with the top one being the application subsystem and the bottom one the supporting hardware. A design specification at one level is said to support the level above by providing a design solution for problems which are formally defined within the level above. This approach is not unlike the one observed in systems which have an onion-like structure with each layer defining a virtual machine. Here the concept is extended to distributed systems, including systems where processes are subject to dynamic reallocation. Design is considered to proceed top down both within levels (though stepwise refinements of design specifications) and between levels in the model.

The top-down development of successive levels in the system halts and the binding phase entered when it becomes obvious that the support needs of the bottom level of the processing model can be implemented by physical hardware structures. In the binding phase, an attempt is made to identify a particular organization of hardware and software components which meets the requirements of the processing model. The designer starts by identifying binding alternatives for the most constrained areas of the specification. This results in the imposition of new constraints over the remaining parts of the design which, in turn, eliminates from consideration many fruitless alternatives. The last task performed is the generation of the software and the hardware requirements for the selected implementation.

System Requirements Definition

SRQ = (CM, R, CQ, eval)

CM - conceptual model
R - domain of possible system realizations
CQ - set of design constraints
eval - system evaluation procedure

The system requirements (SRQ) are defined as a 4-tuple consisting of a conceptual model, a set of possible system realizations, a set of constraints on the system, and a system evaluation function. The conceptual model (CM), which is formally defined below, is a model of the functionality of the system with respect to its environment. The conceptual model does not incorporate any non-functional constraints such as speed or size limitations. This allows for formal treatment of system functions independent of the complexity of performance requirements. The set of system realizations (R) consists of all possible system realizations which exhibit the functionality defined in the conceptual model. Although this set is not necessary for the definition of the system requirements, the concept it represents will be used later in the system design definition. The set of constraints (CQ) consists of all explicit or implied constraints on the performance, packaging, implementation, etc. of the system. The constraints together with the conceptual model fully specify the system to be designed as the conceptual model determines the set of possible realizations and the constraints determine a (possible empty) subset of those

realizations that meet all of the non-functional requirements.

The determination of the subset of realizations which meet all of the requirements is done by the system evaluation function (eval). Given a set of realizations R and a set of constraints C, eval(R,C) is a set containing only those members of R which are consistent with the constraints C. This particular approach has been selected for dealing with impact of constraints because of the difficulty involved in recognizing all classes of active constraints.

Conceptual Model

CM = (ES, ESO, SS, SSO, F)

ES - set of environment states
ESO - set of initial environment states
SS - set of system states
SSO - set of initial system states
F - functionality
F SUBSET.OF ((ES x SS) x (ES x SS))

The conceptual model (CM) is defined as a 5-tuple consisting of a set of environmental states (ES), a set of initial states for the environment (ESO), a set of system states (SS), a set of initial states for the system (SSO), and a transition mapping (F). The sets ES and ESO, where ESO is a subset of ES, are used to model the environment of the system, as the allowable functionality of the environment must be determined in order to establish the functionality of the system. Similarly, the sets SS and SSO, where SSO is a subset of SS, are used to model the system. Given this static characterization of the system and its environment the mapping F is used to model their dynamic properties. F can be described as a subset of ((ES x SS) x (ES x SS)) where the first element in the pair is the current state of the system and the environment, and the second element is the successor state. Since many to many mappings in F are allowed, it is possible to model non-deterministic state transitions. A system/environment interaction sequence ((e1,s1),(e2,s2),...(en,sn),...) is correct iff e1 is a member of ESO, s1 is a member of SSO, and each pair of successive states ((ei,si),(ei+1,si+1)) is a member of F.

Processing Model Definition

PM = (DS1, DS2, ... DSn)

DSi - System design specifications
DS1 IMPLEMENTS SRQ
DSi+1 SUPPORTS DSi for i<n
DSn SPECIFIES MCH

The processing model is defined as a linear order of design specifications which meet a number of criteria. First, the initial design specification (DS1) must implement the system requirements (SRQ). The IMPLEMENTS relationship essentially requires that the design specification accomplish the functionality established by the conceptual model and that at least one realization of that design specification meets the system constraints. The next requirement is that each successive design specification must support its immediate predecessor. The SUPPORTS relationship requires that one design specification implements the support needs (e.g., communication, reallocation, postulated primitive operations, etc.) of another design specification. The last criterion for the processing model is one of completeness: the model completely specifies a system architecture when the lowest level design specification DSn SPECIFIES MCH, where MCH is a net of physical machines. The SPECIFIES relationship is defined more formally later in this paper, but essentially states that certain components of DSn can be mapped directly onto MCH.

System Design Specification

DS = (PSS, PRS, ALC, PFS, BD, C, eval)

PSS - process structure
PRS - processor structure
ALC - process/processor allocation
PFS - performance specifications
BD - binding options
C - constraints
eval - system evaluation procedure w.r.t. C

A system design specification is a 7-tuple consisting of a process structure, a processor structure, an allocation of processes to processors, a set of performance specifications, a set of binding options, a set of system constraints, and a procedure for evaluating systems with respect to the constraints. The process structure is a network of processes and conceptual communications links, and essentially describes the functional elements which interact to carry out the overall system function. The processor structure describes a network of abstract processors and their interconnections which provides the support structure on which the process structure resides. The process/processor allocation defines a mapping of processes and inter-process communications in the process structure onto the processors and interconnections in the processor structure. The performance specifications attach performance requirements to the process and processor structures and also contain performance data derived from these structures that can be used in the design validation process. The binding options represent a conceptual set of feasible realizations of the system design which meet the binding constraints (to be defined later). The set of constraints consists of the constraints established by the system requirements. Finally, the system evaluation function serves to define the set of binding options by evaluating the validity of potential system realizations with respect to a set of constraints. A more formal definition of each of these components of the design specification is presented below.

The system design specification may be considered the skeletal semantic model for a complete distributed systems design language. Unfortunately, no such language is available at this point although specification languages have received considerable attention both in industrial and academic circles. Various proposals range in flavor from tables, standardized document formats, and graphic representations [6], at one extreme, to formal languages having well-defined syntax and semantics [3] at the other. The work on program specifications has largely dominated the field, both with respect to the attention received and level of formality. (The reader is referred to [2] for a good survey of available formal program specification techniques.) The specification of distributed systems, however, continues to present designers with many unresolved problems [2]. The hope is for this work to provide valuable insights that could affect the next generation of distributed systems specification languages.

Process Structure

```
PSS = (PS, LK)

PS - set of processes
     PS = { P | P=(SP, sp0, TP) }
     SP  - set of process states
     sp0 - set of initial process states
     TP  - state transition rule
           TP SUBSET.OF (SP x SP)

LK - set of links between processes
     LK = { L | L=(PL, SL, sl0, TL) }
     PL  - processes linked by L
           PL SUBSET.OF PS
     SL  - states internal to the link
     sl0 - set of initial link states
     TL  - link communication protocol
           TL SUBSET.OF (lcstates x lcstates)
           where
              lcstates = (SL x SP1 x ... SPn)
              Pi MEMBER.OF PL
              Pi = (SPi, sp0i, TPi)
```

The process structure (PSS) is defined as an ordered pair (PS,LK), where PS is a set of processes and LK is a set of links. Processes are the individual functional entities in the system. The set PS describes not only the processes internal to the system, but also the processes in the system environment. Links are the logical communication paths of the system, and can be viewed as the medium by which inter-process message passing is carried out. The combination of the process' behavior with the links' behavior determines the overall functionality of the system and its environment.

A process P is defined as a triple (SP, sp0, TP), where SP is a set of process states, sp0 is a set of initial process states, and TP is a set of valid transitions from one process state to another. The functionality of the process is the set of all valid sequences of states, which can be determined from sp0 through successive application of the transitions in TP (the conceptual model was defined similarly).

A link L is defined as a 4-tuple (PL, SL, sl0, TL), where PL is a set of processes, SL is a set of internal link states, sl0 is a set of initial internal states, and TL is a set of valid transitions which serves to specify the link protocol. PL is a subset of PS, and represents those processes which can communicate through the link subject to the link protocol. The set SL describes the internal states of the link, but does not include knowledge of any portion of the internal process state. The functionality of the link and its interaction with the processes it interconnects is specified by the state transition rules in TL, which specify transitions from one aggregate process/link state to another. Hence, communication is accomplished by the fact that the link has knowledge of and may alter the states of the processes it connects.

Processor Structure

```
PRS = (PR, IC)

PR - set of processors
     PR = { Q | Q=(SQ, sq0, TQ) }
     SQ  - set of processor states
     sq0 - set of initial processor states
     TQ  - state transition rule
           TQ SUBSET.OF (SQ x SQ)

IC - set of processor interconnections
     IC = { W | W=(PW, SW, sw0, TW) }
     PW  - processors being interconnected
     SW  - set of interconnection states
     sw0 - set of initial interconnection states
     TW  - interconnection protocol
           TW SUBSET.OF (icstates x icstates)
           where
              icstate = (SW x SQ1 x ... SQn)
              Qi MEMBER.OF PW
              Qi = (SQi, sq0i, TQi)
```

The processor structure (PRS) is defined as an ordered pair (PR,IC), where PR is a set of processors and IC is a set of processor interconnections. PR is a set of abstract processors, and represent the functional processing elements in the system. IC is a set of processor interconnections, which defines the communications paths between processors. The network of processors and interconnections so described provides the support structure upon which the process structure resides (in a manner like that of a virtual machine supporting the execution of a user process). It also identifies the minimum degree of system distribution required (the distribution may increase in subsequent design specifications). The formal definitions of the processors and interconnections is very similar to that of the processes and links of the process structure, and so the discussion will not be repeated here.

Process/Processor Allocation

```
ALC = (AF, af0, TA)

AF  - set of all allocations functions
   AF = {A | A : prstates --> psstates}
   prstates - composite processor structure
              states
   prstates = SQ1 x SQ2 x ... SQh
                        x SW1 x ... SWk
   psstates - composite process structure
              states
   psstates = SP1 x SP2 x ... SPn
                        x SL1 x ... SLm

af0 - set of initial allocation functions

TA  - set of valid allocation changes
   TA SUBSET.OF
   ((AF x psstates x prstates)x
            (AF x psstates x prstates))

   (!A ((A1,x1,y1),(A2,x2,y2)) ELEMENT.OF TA)
   [ A1(y1)=x1 AND A2(y2)=x2 AND
   ( (A2 = A1 AND x2 = x1 AND TRS(y1,y2))
   OR (A2 = A1 AND TSS(x1,x2) AND TRS(y1,y2))
   OR (NOT(A1=A2) AND x1=x2) )   ]

where
TRS = composite state transition rule for PRS
TSS = composite state transition rule for PSS
```

The process/processor allocation is defined as a triple (AF, af0, TA), where AF is the set of all allocation functions, af0 is a set of initial allocation functions, and TA is a set of transitions between allocation functions. The purpose here is to model the association between processes and processors. Although most systems maintain a static mapping between processes and processors, this model allows for the specification of systems in which the mappings change over time. The valid sequences of transitions is determined from af0 and TA.

The set of allocation functions AF is defined as the set of all mappings from the set of composite processor states (prstates) onto the set of composite process states (psstates). The set prstates is essentially the cross product of all processor and interconnection states in the processor structure, and the psstates is the cross product of the process and link states in the process structure. The mapping is functional, so that a given state of the processor structure uniquely identifies a state of the process structure. Note that it is possible for an individual process state to have a functional dependency on the states of more than one processor, so that a process can effectively be mapped onto more than one processor or interconnection.

The set of valid allocation changes TA is a subset of ((AF x psstates x prstates) x (AF x psstates x prstates)), where each member of an ordered pair in TA represents the association of an allocation function with a specific processor structure state and a process structure state. Each member of TA represents a change in the system allocation function to be associated with a given change in the processor structure and process structure states. Non-determinism is possible as several members of TA may have the same first element but different second elements so that many different changes in the allocation may be possible at a given state of the system. In all cases, the members of TA are subject to the following constraints. First, if the allocation function remains the same across a transition, then a valid change in state of the processor structure must have occurred and the process structure state must have remained the same or undergone a valid transition. Second, if the allocation function changes across a transition then no change in the process structure state must have occurred. These constraints assure that TA is consistent with the transition functions in the processes and processors.

Performance Specifications

```
PFS = (PSRQ, PRRQ, PSCH, PRCH, PMOD)

PSRQ - process structure performance
       requirements
       PSRQ SUBSET.OF (psstates* x PSA)
       PSA - process structure attributes

PRRQ - processor structure performance
       requirements
       PRRQ SUBSET.OF (prstates* x PRA)
       PRA - processor structure attributes

PSCH - process structure performance
       characteristics
       PSCH SUBSET.OF (psstates* x PSA)

PRCH - processor structure performance
       characteristics
       PRCH SUBSET.OF (prstates* x PRA)

PMOD - performance model
       PMOD : POWERSET(prstates* x PRA)
          .      --> POWERSET(psstates* x PSA)

PMOD(PRRQ) ==> PSRQ
PMOD(PRCH) ==> PSCH

PSCH ==> PSRQ
PRCH ==> PRRQ

where
{(x1,y1),...(xn,yn)} ==> {(x1,z1),...(xn,zn)}
IFF (!A i) [ 0<i<n+1 IMPLIES (yi IMPLIES zi)]
```

The performance specification (PFS) is defined as a 5-tuple (PSRQ, PRRQ, PSCH, PRCH, PMOD) where PSRQ is a set of process structure performance requirements, PRRQ is a set of processor structure performance requirements, PSCH is a set of process structure performance characteristics, PRCH is a set of processor structure performance characteristics, and PMOD is a performance model relating the various specifications. The performance requirements

specifications serve to establish performance criteria which must be met by the process and processor structures. The performance characteristics represent derived performance data about the system.

The performance requirements and characteristics are defined as sets of (state sequence, attribute predicate) pairs. The state sequences are linear sequences of 0 or more composite structure states, and represent various subparts of a process or processor structure functionality. The predicates associate specific attributes with each sequence, such as "elapsed time < 100 microseconds". Attributes of the entire structure can be associated with null sequences, such as "system storage < 64K" or "system mass < 40Kg".

The performance model is a function which maps (processor state sequence, attribute) pairs to (process state sequence, attribute) pairs, and represents the derivation of performance requirements or characteristics for a process structure based on a set of requirements or characteristics for a processor structure supporting it. Given the performance model PMOD, then PMOD(PRRQ) must be consistent with PSRQ, and PMOD(PRCH) must be consistent with PSCH, where a requirement R1 is consistent with a requirement R2 iff the attributes associated with sequences in R1 imply the attributes associated with the same sequences in R2. Of course, for the design specification to be valid the performance characteristics in each structure must be consistent with the performance requirements for the structure.

Binding Options

BD = (FB, FR, BC, eval)

FB - feasible bindings
FR - feasible realizations
BC - binding constraints
eval - binding evaluation function

FB = eval(FR, BC)
NOT(eval(FB, C) = nil)

(PSS U PRS U ALC U PSRQ U PRRQ) SUBSET.OF BC

The binding options (BD) are defined as a 4-tuple (FB, FR, BC, eval) where FB is a set of feasible bindings, FR is a set of feasible system realizations, BC is a set of binding constraints, and eval is a binding evaluation function. The set of feasible realizations FR is a conceptual set of all system realizations which meet the functional requirements of the design specification. The set of binding constraints BC represents a cumulative set of constraints which have been imposed on the system binding through the design process. Constraints which are included in BC are such things as binding restrictions derived from the system requirements, restrictions imposed by the choice of process and processor structures at the "current" and all higher levels in the processing model, and restrictions imposed by the performance requirements. The set of feasible bindings FB is the set of all feasible realizations which are consistent with the binding constraints, i.e. FB = eval(FR,BC). For the system design to be valid, it must also be true that eval(BD,C) is not empty, where C is the set of constraints from the design specification. Hence, there must be at least one binding of the design to a system realization which meets all of the system constraints.

Hardware/Software Requirements Definition

HSRQ = (SFRQ1,...,SFRQn,HDRQ)

SFRQi - subsystem software requirements
HDRQ - hardware requirements

The hardware/software requirements specification is produced by the binding phase and is the final product of the system design process. The list of specifications SFRQ1 ... SFRQn+1 are requirements specifications for software systems required for each of the n levels of the processing model. The functional and performance requirements for the software at each level are derived from the process structure of its corresponding level in the processing model. The hardware requirements specification HDRQ is derived from the processor structure of the design specification DSn. The functional and performance specifications for the hardware are thus derived directly from the processor structure. Additional elements of the specifications include the identification of existing hardware and software components, the binding of process and processor structure elements to the existing components, and a specification of the interfaces required between components.

Relationships Between Specifications

There are four basic relationships which tie together the specifications of the information structures in the processing model. The first relationship, IMPLEMENTS, defines the requirements that must be met for a design specification of a system to be consistent with the system requirements definition. The SUPPORTS relationship ties a design specification to another design specification which provides the support structure for it. The REFINES relationship describes the relationship between design specifications in the stepwise refinement of a design at one level in the processing model. Finally, the SPECIFIES relationship ties a design specification to a network of physical machines.

Implements

```
DS = (PSS, PRS, PFS, ALC,
                    BD=(FB, FR, BC, eval), C)
SRQ = (CM=(ES, ES0, SS, SS0, F), R, CQ, eval)

DS IMPLEMENTS SRQ IFF

(?E PHI) [ PHI: psstates --> (ES x SS) ]
    where
    PHI is onto  (ES x SS)
    TSS(x1, x2) AND NOT( PHI(x1) = PHI(x2) )
       IMPLIES  (PHI(x1), PHI(x2)) MEMBER.OF F

FR = R
C = CQ
```

The implements relationship is essentially a predicate asserting that a design specification specifies at least one system realization that meets a system requirements specification. The relationship is established if there exists a mapping of composite process structure states in the design specification onto the composite system-environment state set in the system requirements which has the following properties. First, the constraints CQ in SRQ must be the same as the constraints C in DS. Second, the set FR of feasible realizations in DS must be the same as the set R of valid system realizations in SRQ. Finally, any valid transition between states in DS must map onto either no change in states of the conceptual model or a valid change of states of CM. Hence, PHI represents a mapping of the functionality of DS onto the functionality of SRQ, with DS and SRQ required to have the same set of constraints.

Supports

```
DS  = (PSS, PRS, ALC, PFS, BD, C, eval)
DS' = (PSS', PRS', ALC', PFS', BD', C', eval)

DS' SUPPORTS DS IFF

(?E PSI) [PSI : (PS' U LK') --> (PR U IC)
    PSI is onto (PR U IC)
    IF (Pi' MEMBER.OF PS') AND
       (PSI(Pi') MEMBER.OF IC)
       THEN (!A Lj' MEMBER.OF LK')
         [IF Pi' MEMBER.OF PLj'
          THEN PSI(Pi') = PSI(Lj')]
    IF (Lj' MEMBER.OF LK') AND
       (PSI(Lj') MEMBER.OF PR)
       THEN (!A Pk' MEMBER.OF PLj')
         [(PSI(Pk') = PSI(Lj'))]   ]

(?E QSI) [QSI : (PR' U IC') --> (PR U IC)
    QSI is onto (PR U IC)
    IF (Qi' MEMBER.OF PR') AND
       (QSI(Qi') MEMBER.OF IC)
       THEN (!A Wj' MEMBER.OF IC')
         [IF Qi' MEMBER.OF PWj'
          THEN QSI(Qi') = QSI(Wj')]
    IF (Wj' MEMBER.OF IC') AND
       (QSI(Wj') MEMBER.OF PR)
       THEN (!A Qk' MEMBER.OF PWj')
         [(QSI(Qk') = QSI(Wj'))]   ]

(?E PHI) [(PHI : psstates' --> prstates)
    PHI is onto prstates
    IF TSS'(x1', x2') AND
       NOT(PHI(x1')=PHI(x2'))
       THEN TRS(PHI(x1'), PHI(x2')) ]

(!A Qi MEMBER.OF PR) (?E PHI1.i)
    [(PHI1.i : psstates.i' --> SQi)
    PHI1.i is onto SQi
    IF TSS.i'(x1', x2') AND
       NOT(PHI1.i(x1')=PHI1.i(x2'))
       THEN TQi(PHI1.i(x1'), PHI1.i(x2')) ]

(!A Wj MEMBER.OF IC) (?E PHI2.j)
    [(PHI2.j : psstates.j' --> SWj)
    PHI2.j is onto SWj
    IF TSS.j'(x1', x2') AND
       NOT(PHI2.j(x1')=PHI2.j(x2'))
       THEN TWj(PHI2.j(x1'), PHI2.j(x2')) ]

AF' = { A' | A' : prstates' --> psstates' AND
    A' = (Aq1',...,Aqn', Aw1',...,Awm')   AND
    (!A Qi MEMBER.OF PR)
      [ Aqi : prstate.i' --> psstate.i' ] AND
    (!A Wj MEMBER.OF IC)
      [ Awj : prstate.j' --> psstate.j' ] }

where
psstate.i' - composite state of entities
             mapped by PSI into Qi
TSS.i'     - corresponding state transition
             rule
prstate.i' - composite state of entities
             mapped by QSI into Qi
TRS.i'     - corresponding state transition
             rule
psstate.j' - composite state of entities
             mapped by PSI into Wj
TSS.j'     - corresponding state transition
             rule
prstate.j' - composite state of entities
             mapped by QSI into Wj
TRS.j'     - corresponding state transition
             rule

(?E KAI) [KAI : (psstate'* x PSA') -
                        --> (prstate* x PRA))
    KAI(PSRQ') = PRRQ
    KAI(PSCH') = PRCH ]

FB' = eval(FR', BC')
    FR' SUBSET.OF FR
    BC  SUBSET.OF BC'

C' = C
```

The SUPPORTS relationship between two design specifications DS' and DS states that DS' specifies the support system upon which DS resides, or more precisely that DS' is a design specification for the processor structure of DS. A number of requirements must be met for the relationship to hold, and these are described below. The first set of requirements deals with the mapping of processes, processors, links, and interconnections in DS' to processors and interconnections in DS. For processes and links,

there must exist a function PSI from processes and links in DS' onto processors and interconnections in DS. Under PSI, if any process in DS' maps to an interconnection in DS, then all links to which that process is attached in DS' must map to the same interconnection in DS. Similarly, if any link in DS' maps to a processor in DS then all processes connected by the link must also map to the same processor. For processors and interconnections in DS', there must exist a function QSI from processors and interconnections in DS' to processors and interconnections in DS. Under QSI, if any processor in DS' maps to an interconnection in DS, then all interconnections to which that processor is attached in DS' must also map to the same interconnection in DS. Similarly, if an interconnection in DS' maps to a processor in DS then all processors connected by that interconnection must map to the same processor in DS. The result of these restrictions is that the processes, processors, links, and interconnections of DS' are partitioned into disjoint sets according to the processor or interconnection in DS to which they are mapped. In other words, DS' may increase the level of distribution identified by DS, but only in a manner consistent with DS.

The second group of restrictions deals with the functional mapping between the two design specifications. In a global sense, there must exist a mapping PHI from the composite process structure state (psstates') in DS' to the composite processor structure state (prstates) in DS such that valid transitions in psstates' map to either valid transitions in prstates or no transition. Using the partitioning from above, however, we can qualify the mapping further. For every processor Qi in the processor structure of DS, we can define a composite process state psstates.i' which is the composite state of all links and processes in DS' which map to Qi through PSI. Similarly, for each interconnection Wj in DS we can define a composite process state psstates.j'. We can now define a mapping between the partitions of the processing model in DS' and the processors and interconnections in DS as follows. For each processor Qi in DS, there must exist a mapping PHI1.i from psstates.i' onto SQi such that all valid transitions in psstates.i' map to valid transitions in SQi or no transition in SQi. Also, for each interconnection Wj in DS there must exist a mapping PHI2.j from psstates.j' onto SWj such that all valid transitions in psstates.j' map to valid transitions in SWj or no transition.

Using the partitioning concept a restriction can also be placed on the the process/processor allocation in DS'. The requirement is essentially that processes within a partition mapping to a processor Q in DS can only be allocated to processors in DS' which are also mapped to Q. Put more formally, let us define prstates.i' as the composite state of the processors and interconnections in DS' which map through QSI to processor Qi in DS, and prstates.j' be the composite state of the processors and interconnections which map through QSI to interconnection Wj in DS. Then the allocation functions A' in DS' can be decomposed into subfunctions Aq1',...Aqn',Aw1',...Awm' such that the Aqi map prstates.i' onto psstates.i' and the Awj map prstates.j' onto psstates.j', subject to the state transition consistency rules for allocation functions.

The last set of requirements deals with the constraints and performance requirements and characteristics of the two models. First, there must exist a function KAI which maps (psstates sequences, attribute predicate) pairs onto (prstates sequences, attribute predicate) pairs such that KAI(PSRQ') is consistent with PRRQ and KAI(PSCH') is equal to PRCH. Of course, the mapping of psstates' sequences to prstates sequences implied by KAI must be consistent with the function PHI. In this way KAI provides the translation between the performance requirements and characteristics in DS' and those in DS. Second, the set of constraints C' in DS' must equal the set of constraints C in DS. Finally, FR' must be a subset of FR and BC a subset of BC'. The latter condition results as BC' contains the binding constraints imposed by the design decisions in DS as well as those imposed by design decisions in DS'.

Refines

```
DS   = (PSS, PRS, ALC, PFS, BD, C, eval)
DS'  = (PSS', PRS', ALC', PFS', BD', C', eval)

DS' REFINES DS IFF

(?E PHI) [PHI : psstates' --> psstates
    PHI is onto psstates
    IF TSS'(x1', x2') AND
       NOT(PHI(x1')=PHI(x2'))
       THEN TSS(PHI(x1'), PHI(x2')) ]

(?E QSI) [QSI : prstates' --> prstates
    QSI is onto prstates
    IF TRS'(y1', y2') AND
       NOT(QSI(y1')=QSI(y2'))
       THEN TRS(QSI(y1'), QSI(y2')) ]

(!A  A MEMBER.OF AF)
     [ x = (x1,...,xn,...,xn+m)
     y = (y1,...,yn,...,yn+m)
     with  yi MEMBER.OF SQi for 0<i<n+1
           yi MEMBER.OF SWi for n<i<n+m+1

     A = (A1,...,An,...,An+m)
     with  Ai(yi) = xi   for 0<i<n+m+1  ]

(!A  A' MEMBER.OF AF')
     [ x' = (x11',...,x1k(1)',...,
               xn1',...,xnk(n)',...,
               x(n+m)1',...,x(n+m)k(n+m)')

     y' = (y11',...,y1k(1)',...,
             yn1',...,ynk(n)',...,
             y(n+m)1',...,y(n+m)k(n+m)')
```

```
        with
           yij' MEMBER.OF SQij'
                    for 0<i<n+1   and 0<j<k(i)
           yij' MEMBER.OF SWij'
                    for n<i<n+m+1 and 0<j<k(i)

        A' = (A11',...,A1k(1)',...,
              An1',...,Ank(n)',...,
              A(n+m)1',...,A(n+m)k(n+m)')
             with Aij'(yij') = xij'
                  for 0<i<n+m+1 and 0<j<k(i)

        PHI(..., xi1',...,xik(i)', ...)
             = (..., xi, ...)

        QSI(..., yi1',...,yik(i)', ...)
             = (..., yi, ...) ]

    (?E KHI) [ KHI : (psstate'* x PSA')
                          --> (psstate* x PSA)
        KHI(PSRQ') ==> PSRQ
        KHI(PSCH') = PSCH  ]
    (?E KSI) [ KSI : (prstate'* x PRA')
                          --> (prstate* x PRA)
        KSI(PRRQ') ==> PRRQ
        KSI(PRCH') = PRCH  ]

    FB' = eval(FR', BC')
        FR' SUBSET.OF FR
        BC  SUBSET.OF BC'

    C' = C
```

During the process of system design it is usually necessary to decompose a design specification at one level of the processing model into a more detailed specification of the same level. It is therefore necessary to define a relationship REFINES between a design specification DS and its refinement DS' in order to formalize this notion of decomposition. In contrast with current practices, our definition of stepwise refinement allows for concurrent refinement of both the process and processor structures. This extension, termed dual stepwise refinement, is particularly important in the design of high performance systems where hardware and software designs may not be carried out independently. For the relationship DS' REFINES DS to be valid, the following requirements must hold.

The first requirement is that there be a functional mapping between the two design specifications. For the process structures, there must exist a function PHI from the composite process structure state of DS' to that of DS, such that valid transitions in the process structure state of DS' map onto valid process structure state transitions in DS, or no transition. A similar function QSI from the composite processor structure state of DS' to that of DS must also exist.

The second requirement is that the allocation functions of the two design specifications must be consistent with the functional mappings PHI and QSI. Let the allocation functions A in DS be broken up into n+m sub-allocations A1, A2, ...An+m, where n is the number of processors in PR and m is the number of interconnections in IC. If yi is the state of a single processor or interconnection, then xi = Ai(yi) is the composite state of all processes and links which are allocated at least in part to the given processor or interconnection. In the refinement DS', there must be corresponding allocations Aij' and states xij', yij' such that Aij'(yij') = xij', PHI(...xi1',...xik(i)'...) = (...xi...) and PHI(...yi1',...yik(i)...) = (...yi...). In other words, if a set of processes (or links) ps in DS is allocated to a specific processor (or interconnection) pr in DS, then the set of processes (links) ps' in DS' onto which ps is refined must be allocated within the set pr' of processors (interconnections) which correspond to pr in the refinement. Aside from the restriction to the allocation function, this requires that processors and interconnections be decomposed in a hierarchical manner.

The final set of requirements deals with the consistency of the binding options and performance requirements. For process structures there must exist a mapping KHI from (process state sequence, attribute predicate) pairs in DS' to (process state sequence, attribute predicate) pairs in DS such that KHI(PSRQ') is consistent with PSRQ and KHI(PSCH') is equal to PSCH. For processor structures, there must exist a similar mapping KSI between (processor state sequence, attribute predicate) pairs in DS' and DS such that KSI(PRRQ') is consistent with PRRQ and KSI(PRCH') is equal to PRCH. Next, the set of constraints C' in DS' must equal the set of constraints C in DS. Finally, the set of feasible realizations FR' in DS' must be a subset of FR, and BC' a superset of BC.

Specifies

```
    DS   = (PSS, PRS, ALC, PFS, BD, C, eval)
    MCH  = (PSS", PRS", ALC", PFS", BD", C", eval)

    DS SPECIFIES MCH IFF

       MCH SUPPORTS DS
       QSI is one-to-one
       SOFTWARE(PSS")
       HARDWARE(PRS")
       STATIC(ALC")
       i.e.,
       TA" = (({A"} x psstates" x prstates")**2)
```

A processing model is in a sense complete when its lowest level design specification can be mapped onto a physical set of hardware and software. Given a machine level specification MCH and a design specification DS, we say that DS SPECIFIES MCH if the following requirements are met. First, the machine level specification MCH must have its processor structure map directly to a hardware implementation and its process structure map directly to a software implementation on the given hardware. Second, the process/processor allocation in MCH must be

static. Third, the relationship MCH SUPPORTS DS must be valid. Finally, the function QSI mapping processors and interconnections in MCH to processors and interconnections in DS must be one to one.

Conclusions

Although our work has been motivated by the nature of a particular methodology, the information structures and the consistency constraints are not specific to a single methodology. There are two reason for this. First, an attempt has been made to maintain the greatest possible degree of generality in all definitions. Second, the nature of the information structures and consistency constraints is independent of the order in which they are constructed and, consequently, they may be shared by methodologies that employ quite distinct design strategies. For instance, although the processing model is structured in a top-down hierarchical manner, since it is merely an interface between the architecture design phase and the binding phase it does not preclude the use of a bottom-up oriented design methodology within the system architecture design phase. Neither is it necessary that the processing model be completed before binding is undertaken. The formalization of the concepts of hierarchical structuring of a distributed system and dual stepwise refinement of process and processor structures represents an important contribution to a better understanding of distributed system design.

References

[1] Alford, M., "Requirements for Distributed Data Processing Design," Proc. 1'st Int. Conf. on Distributed Computing Systems, pp. 1-14, October 1979.

[2] Liskov, B. and Berzins, V., "An Appraisal of Program Specifications," Research Directions in Software Technology, P. Wegner, editor, MIT Press, pp. 276-301, 1979.

[3] Robinson, L. and Levitt, K. N., "Proof Techniques for Hierarchically Structured Programs," CACM 20, No. 4, pp. 271-404, April 1977.

[4] Roman, G.-C., "On Reducing Ambiguities in Methodology Definitions," Proc. of 1982 Conf. on Trends and Applications, pp. 47-54, May 1982.

[5] Roman, G.-C., Stucki, M. J., and Gillett, W., "The Total System Design (TSD) Methodology: From Problem Definition to Hardware/Software Requirements," Technical Report No. WUCS-82-4, Department of Computer Science, Washington University, 1982.

[6] Ross, D. T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Trans. on Soft. Eng. SE-3, No. 1, pp. 16-34, January 1977.

[7] Rustin, R., Formal Semantics of Programming Languages, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[8] Wymore, A. W., A Mathematical Theory of Systems Engineering - the Elements, John Wiley and Sons, New York, 1967.

Summary Of Notation

n-tuple	(a, b, ...)
set definition	{a, b, ...}
	{x ¦ predicate(x)}
function definition	fname: domain --> range
quantifiers	
existential	(?E x) [predicate]
universal	(!A x) [predicate]
cross product	set1 x set2
n'power cross product	set**n
logical implication	IF pred1 THEN pred2
	pred1 IMPLIES pred2
if and only if	IFF
logical operators	AND, OR, NOT
subset test	set1 SUBSET.OF set2
set membership test	element MEMBER.OF set
set operations	UNION, INTERSECTION, MINUS
set of all sequences over some set	set*
set of all subsets	POWERSET(set)
empty set	nil