On the Construction of Submodule Specifications and Communication Protocols

PHILIP MERLIN
The Technion
GREGOR V. BOCHMANN
University of Montreal

The problem of elaborating the specification for the submodules of a system is considered. A new method for the construction of submodule specifications is described. If the system is to consist of n submodules and the system as well as $(n - 1)$ submodules are specified, then the method described determines the specification of the additional nth submodule. A formula is given which defines the specification of the additional submodule in the general case where module specifications are given in terms of sets of possible execution sequences, and interaction occurs when several modules participate in the execution of an atomic interaction. For the restricted context of finite-state machines, a constructive algorithm for the evaluation of the formula is given. The use of this design method is demonstrated by examples, including a simple communication protocol involving error detection and retransmission. Possible applications in other areas, as well as remaining problems, are indicated.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols; D.2.1 [**Software Engineering**]: Requirements/Specifications—*methodologies*; D.2.2 [**Software Engineering**]: Tools and Techniques—*modules and interfaces*; D.4.4 [**Operating Systems**]: Communications Management

General Terms: Design, Languages

Additional Key Words and Phrases: Communication protocols, distributed system design, module decomposition, stepwise refinement

I wish to dedicate this paper to my friend and colleague Philip Merlin. The work described here was done shortly before his unexpected death. In writing this paper, I remember the discussions we had on this exciting topic. I respect Philip Merlin very much and hope that this paper lives up to his expectations.

1. INTRODUCTION

The problem of designing systems with parallel processes has drawn much attention. Systems for distributed processing, data communications, process control, and general operating systems are considered. Because of the complexity

The work described here was performed at the University of Montreal and was partially supported by the Natural Sciences and Engineering Council of Canada. This paper was written during a sabbatical leave at Stanford University partially supported by the Advanced Research Projects Agency under contract MDA903-79-C-0680.

Authors' addresses: P. Merlin, deceased; G. V. Bochmann, Département d'informatique et de recherche opérationnelle, Université de Montréal, Case postale 6128, Succursale "A," Montréal, Québec H3C 3J7, Canada.

of many of these systems, good design methods are important. The usual approach to the design of parallel systems involves the important step of dividing the overall system into a number of separate submodules which operate in parallel and interact in some well-defined way. Usually, this step is repeated for each of the submodules so obtained, which leads to a stepwise refinement of the system specification.

At each step, a given module is subdivided into several submodules that together satisfy the specification of the given module. Usually, the design of the submodules and the determination of their specification is completely left to the designer of the system. If the specifications are sufficiently formal, he may be assisted by some automated system in verifying that the submodule specifications conform to the specification of the given module.

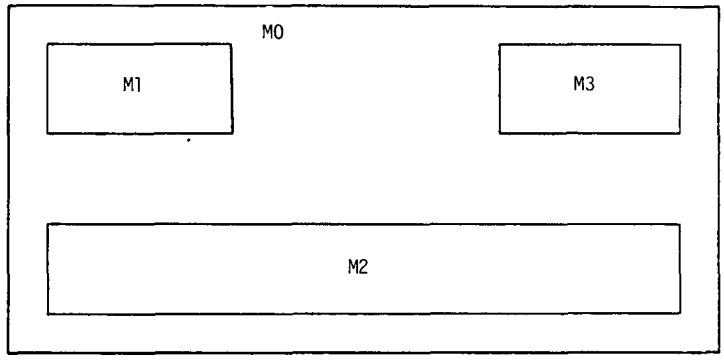
This paper presents a new approach which assists in the elaboration of the submodule specifications during each single step of the refinement process. Given a complete specification of a given module and the specifications of some submodules, the method described below provides the specification of an additional submodule that, together with the other submodules, will provide a system that satisfies the specification of the given module (if such an additional submodule is possible). This is indicated in Figure 1a, which shows the construction of a module M_0 (the system) out of three submodules M_1, M_2, and M_3. If the specifications for M_0, M_1, and M_2 are given, then the specification for M_3 is found by our method.

We think that this approach is useful for the design of a distributed system. We note that the method finds the most general specification possible for the additional submodule. The existing specifications may also be checked for consistency. If there are any inconsistencies, no appropriate additional submodule exists, and the submodule determined by our method will not provide, together with the existing submodules, all the aspects of behavior specified for the system. The method should be complemented with a method for determining whether all specified behaviors are realized and for detecting possible deadlocks and useless operations that may be included in the general specification obtained for the additional submodule.

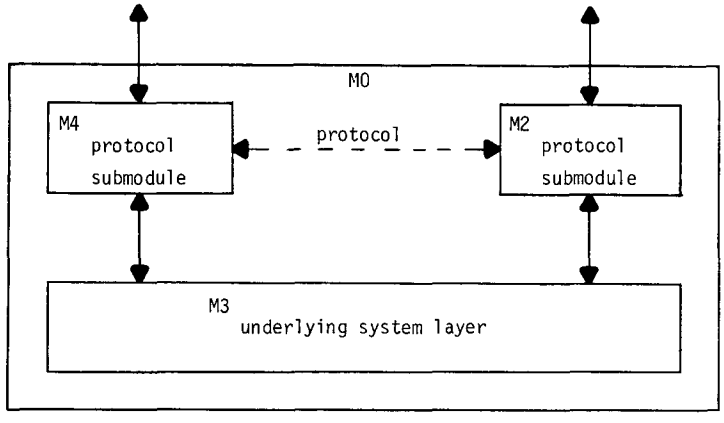
A particular application of this method lies in the design of communication protocols. In this case, we assume that the specifications for the communication services to be provided by the protocol and by the underlying system layer, respectively, are given. We also assume that the specification for one of the protocol submodules is given. Then our method provides a specification for the other protocol submodule automatically (see Figure 1b).

The construction method described here is independent of any particular specification language that may be used for the specification of the modules and submodules. The method is based on a model of parallel systems, described in Section 2, in which a module specification is given as a set of possible execution sequences. For practical purposes, a language is needed for specifying such sets of sequences. Different specification languages may be used. For demonstrating the ideas, we use in this paper a finite-state- (FS-)oriented specification language.

Related work is described by Cerny and Marin [6, 7], who apply a similar method to the design and testing of logical circuits, and by Zafiropulo et al. [16], who describe an interactive protocol design method. In contrast to ours, however,



(a)



(b)

Fig. 1. (a) The structure of a module (system) M_0 consisting of three submodules. (b) The structure of a communication subsystem involving a communication protocol.

their method assumes that all "sending" actions of *both* interacting modules are provided by the designer, while the "receiving" actions are found automatically. No checks against specific service specifications are made.

2. A MODEL FOR SPECIFYING MODULES AND THEIR INTERACTION

This section gives an informal explanation of the descriptive model for system and submodule specifications used in this paper. Section 3 describes the method for constructing submodule specifications and explains its results. A formal description of this topic and larger examples are given in the following sections.

2.1 Module Specification

We assume that a module M_i is characterized by a certain set V_i of externally visible "operations" that are executed by the module. The activity of the module will result in some particular execution sequence over these operations. The module specification determines the set of all possible execution sequences, which

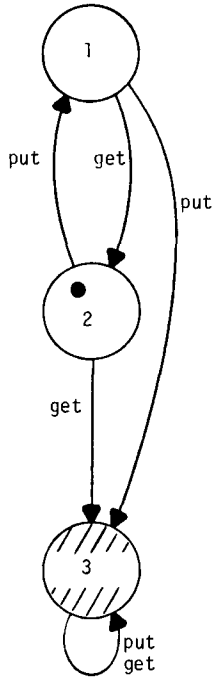


Fig. 2. Transition diagram for a *BUFFER* module. (Note: The shaded state is nonaccepting.)

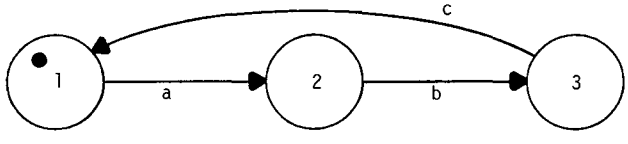


Fig. 3. Transition diagram for a 3-*CYCLE* module. (Note: Nonaccepting states and the transitions leading to them are not shown.)

we write S_i. We note that nothing is assumed about the internal structure of the module or the manner in which the specified execution sequences may be realized.

These assumptions are consistent with many specification methods described in the literature, such as Parnas' information hiding or the use of abstract data types. As far as a given module is concerned, its operations are considered atomic actions, in the sense that they may be assumed to have no extension in time. We note that the overlapping in time of two simultaneous actions may be modeled by considering the beginning and the termination of these actions as atomic. Our model is particularly related to specification methods based on execution sequences, such as those described in [1, 5, 11].

We demonstrate the concepts explained in this paper with examples based on the modules defined in Figures 2 and 3. We use a finite-state specification language here, but other specification languages may be used with our constructive approach. Figure 2 shows a transition diagram for an FS machine defining the possible operation sequences of a *BUFFER* module. The possible operations

are *PUT* and *GET*. The possible operation sequences of the module are those sequences for which the diagram provides a path to an accepting state. The resulting behavior may also be characterized by the regular expression (*PUT GET*)*. It is the behavior of a queue of maximum length one. Figure 3 shows the transition diagram for a 3-*CYCLE* module which executes the events *A*, *B*, and *C* in cyclic order. Transitions to nonaccepting states, similar to those in Figure 2, are not shown but are implied by Figure 3.

2.2 Module Interaction

We now consider several modules running in parallel. In the case of no interaction between the modules, the possible execution sequences of the overall system are obtained by arbitrary interleaving (shuffling) of the execution sequences of the individual modules. Interaction is introduced by requiring that certain operations of different modules be executed jointly. The joint execution of such a group of operations is considered an atomic (indivisible) action. An operation that is part of such an action cannot be executed alone, but only in conjunction with the other operations. For such a group of operations to occur, it is necessary that each operation be possible according to the specification of the respective module. We say that the operations of such an action are "directly coupled" [3] and that the operations or the respective modules "participate" in the action. The joint execution of the directly coupled operations may then be called an "interaction."

We note the consistency of these definitions with many existing methods for specifying module interactions.

(1) For example, the operations of calling a procedure of a different module and the execution of that procedure by that module may be considered directly coupled operations, provided that the execution of the procedure may be considered an atomic action.

(2) Another example is given by the exchange of messages between modules. The events of sending and receiving may be considered directly coupled operations if the delay between the sending and the receiving may be ignored. This is, for instance, the case when the receiver or the sender always waits for the other to be ready [8]. In the case that the message exchange delay between modules is not negligible, collisions between messages sent in opposite directions may occur. Discussion of such collisions is outside the scope of this paper (see, for example, [16]). We note that under certain regularity conditions [2] the delays have no influence on the overall behavior of the system. Collisions may also be modeled by a particular submodule that represents the "message transmission medium" to which the other modules are directly coupled.

(3) The concept of direct coupling also fits into the framework of hardware specification, where the setting of a given circuit to a particular value by one module and the sensing of this value by another module may be considered directly coupled operations.

We believe that the simplicity of this concept (symmetry between the interacting modules, and close synchronization) makes it particularly suitable for many applications, including the submodule construction method discussed below. Formal models for similar module interactions are described in [12, 13].

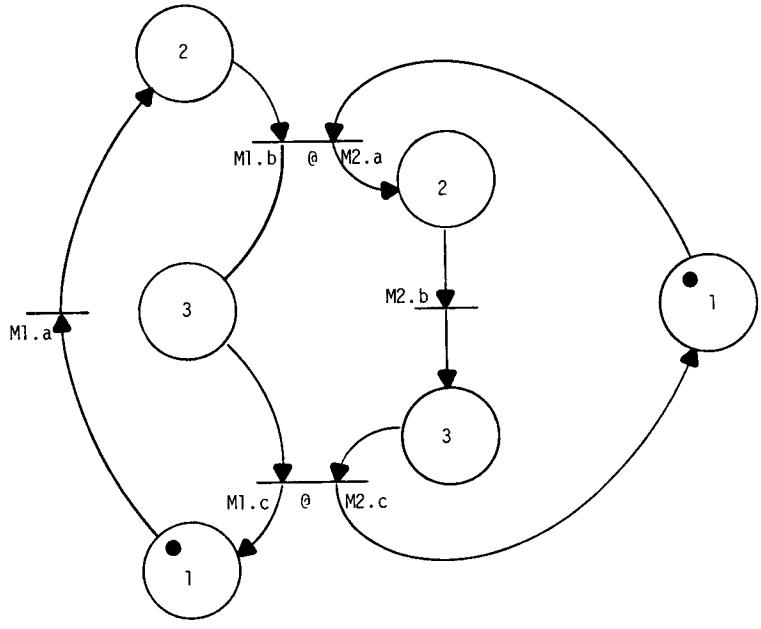


Fig. 4. Petri net description of two directly coupled 3-*CYCLE* modules.

In order to describe which operations of different modules are directly coupled, one needs some "connection specification language" (see, for example, [10]). We use in the following the notation "$A@B$" to indicate that the operations A and B are directly coupled. We write "$M_i \times M_j$" for the system consisting of the two modules M_i and M_j that interact through the direct coupling of some of their operations.

As an example, we consider two modules M_1 and M_2 which are both 3-*CYCLE* modules. They interact through the direct coupling of the operations $M_1.C$ with $M_2.C$ and $M_1.B$ with $M_2.A$ (where "$M_1.C$" stands for "the operation C of module M_1," etc.). The resulting system has the atomic actions $M_1.A$, $(M_1.B@M_2.A)$, $(M_1.C@M_2.C)$, and $M_2.B$. The specification of the 3-*CYCLE* module of Figure 3 implies that these operations must be executed cyclically as defined by the regular expression $(M_1.A\ (M_1.B@M_2.A)\ (M_1.C@M_2.C)\ M_2.B)^*$. A system description in the form of a Petri net is shown in Figure 4. This example shows how directly coupled finite-state machines may be modeled by Petri nets, a particular case of Prinoth's method for describing multiprocess systems [14].

Another example is given by considering two modules M_1 and M_2 that are a *BUFFER* and a 3-*CYCLE* module, respectively. They interact through the direct coupling of the events *PUT* and A. The atomic actions of the resulting system are *GET*, *PUT* A, B, and C. Given the specification of M_1 and M_2 in Figures 2 and 3, the possible operation sequences of the system are shown in Figure 5, where the symbol "X" stands for "$PUT@A$." This example exhibits some form of nondeterminism due to different possible interleavings of operations.

Figure 5 indicates how a specification of a coupled system may be obtained in the case that the specifications are given in a FS-oriented language. Given two FS machines specifying the possible operation sequences of the modules M_1 and

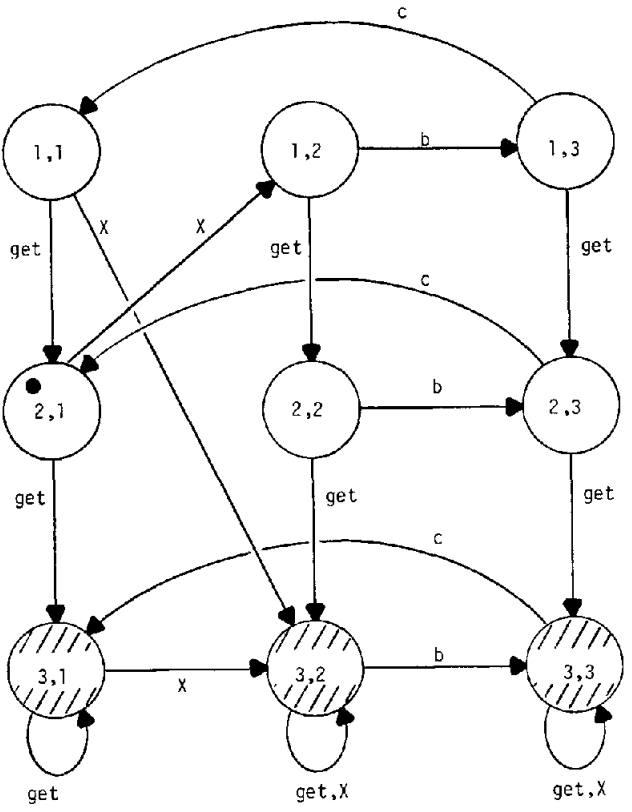


Fig. 5. Transition diagram for a system consisting of directly coupled *BUFFER* and 3-*CYCLE* modules. (Note: $\langle i, j \rangle$ represents a system state where the *BUFFER* module is in state i and the 3-*CYCLE* module is in state j; X stands for the coupled transition.)

M_2 separately, an FS machine M_0 specifying the possible operation sequences of the modules directly coupled is constructed as follows:

(1) The states of M_0 have the form $\langle z_1, z_2 \rangle$ where z_1 and z_2 are states of M_1 and M_2, respectively.

(2) There is a transition from $\langle z_1, z_2 \rangle$ to $\langle z_1', z_2' \rangle$ on the input operation e if

a. e is uncoupled and in V_1, and there is an "e" transition from z_1 to z_1' in M_1, and $z_2 = z_2'$;
b. e is uncoupled and in V_2, and there is an "e" transition from z_2 to z_2' in M_2, and $z_1 = z_1'$; or
c. the operation is coupled and of the form $M_1.e_1@M_2.e_2$, and there is an "e_1" transition from z_1 to z_1' in M_1 and an "e_2" transition from z_2 to z_2' in M_2.

(3) A state $\langle z_1, z_2 \rangle$ is accepting if both states z_1 and z_2 are accepting in their respective FS machines.

Applied to the example of two coupled 3-*CYCLE* modules considered above, this algorithm yields the transition diagram of Figure 6, which shows the cyclic execution order mentioned above.

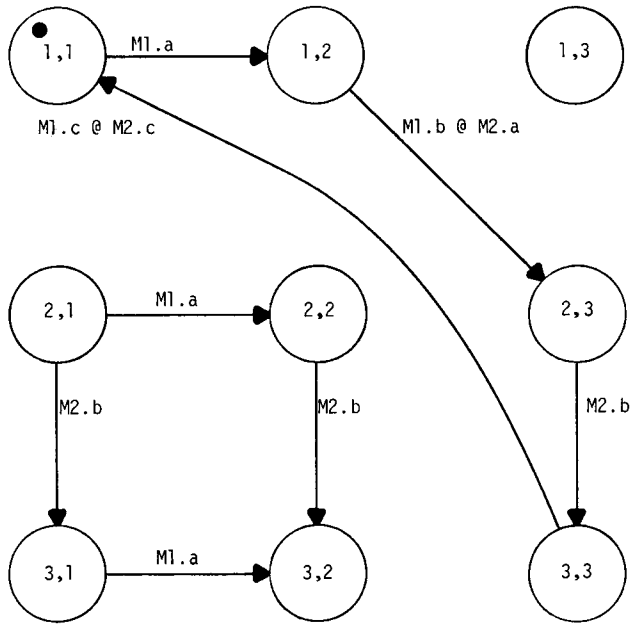


Fig. 6. Transition diagram for a system consisting of two directly coupled 3-*CYCLE* modules.

2.3 Abstractions

Since a group of directly coupled operations are considered an atomic action, we do not need to distinguish among them, or among the contributions of the different participating modules. To simplify the following discussions we assume that a renaming has been performed such that all atomic actions in the system have distinct names. We assume a vocabulary V of names, where each name identifies exactly one atomic action in the system, that is, either a particular uncoupled operation of a particular module, or a particular group of directly coupled operations of a certain set of interacting modules. For example, in the case of the two coupled 3-*CYCLE* modules considered above, the following renamings may be applied:

$$M_1.A \text{ becomes } A;$$
$$M_1.B@M_2.A \text{ becomes } D;$$
$$M_2.B \text{ becomes } B;$$
$$M_1.C@M_2.C \text{ becomes } E.$$

The resulting set of atomic actions is $V = {}_1\{A, D, B, E\}$.

In the following we write V_i for the set of atomic actions in which the module M_i participates. In the example above, for instance, $V_1 = \{A, D, E\}$ and $V_2 = \{D, B, E\}$. Clearly, we always have that $V = \bigcup_i V_i$ and that $V_i \cap V_j$ is the set of atomic actions in which the modules M_i and M_j (and possibly others) participate.

While the process of stepwise refinement leads to more and more detailed specifications, involving an increasing number of different modules, operations, and atomic actions, the verification of a refinement step involves the opposite,

namely, abstracting from the submodule structure of a given module and ignoring all those operations of the submodules that are not visible at the level of the module specification. In the framework of our model, abstraction is obtained by projections, as explained below.

Given a specification of a system of several interacting modules involving a certain set V of atomic actions and a subset of actions considered "relevant," the "projection onto the relevant actions" of the specification is obtained by deleting in each of the possible execution sequences the actions that are not relevant. In the following, we often consider the projection for which the relevant actions are those in which a given module M_i participates, that is, V_i. We write "p_i" for this projection.

As an example, we consider again the two coupled 3-*CYCLE* modules. Applying the projection p_1 yields the specification $(A\ D\ E)^*$, which is isomorphic (i.e., equal except for renaming) to M_1 (which must be so, of course). Applying a projection keeping as relevant actions only A and B yields the specification $(A\ B)^*$, which is isomorphic to a *BUFFER* module. This shows that a *BUFFER* module may be implemented by two interacting 3-*CYCLE* modules. We note that in this case the "interactions" between the two modules are considered irrelevant, but, clearly, their existence is a determining factor for the behavior of the system.

In the case of the FS specification language, which we use for our examples, the projection of a specification is performed by deleting the irrelevant action symbols in the regular expression or by replacing, within the transition diagram of the module, all transitions on irrelevant actions by a spontaneous transition, and subsequently simplifying the diagram, if necessary.

3. CONSTRUCTION OF SUBMODULE SPECIFICATIONS

We now explain a method for constructing submodule specifications. For simplicity, we assume the case of a system made up of two submodules. The general case can be reduced to this case, as shown in Section 4.

We want to solve the following problem: Given the specification of a module or system M_0 and of one of the submodules, say M_1, we want to find a submodule M_2 such that the system consisting of the interacting submodules M_1 and M_2 satisfies the specification for M_0. As far as the operations of the different modules are concerned, the interaction model described above implies the following: Let V_0, V_1, and V_2 be the sets of atomic actions in which the modules M_0, M_1, and M_2, respectively, participate. (We assume that the correspondence between these actions and the operations of M_0 and M_1 is given.) V_0 and V_1 are determined from the specifications of M_0 and M_1. It is clear that those actions of V_0 that are not in V_1 must be in V_2, because otherwise they could never be executed by the interacting modules M_1 and M_2. Also, V_2 must include some actions of V_1, because otherwise there would be no interaction between the two modules. On the other hand, any action of M_2 in which neither M_0 nor M_1 participates is not relevant to our considerations, because we may abstract from it without affecting the possible interactions of the module M_2. Therefore, if we exclude the possibility of actions in which more than two modules participate, we may deduce the set V_2 of actions in which the unknown module M_2 participates to be

$$V_2 = (V_0 - V_1) \cup (V_1 - V_0)$$

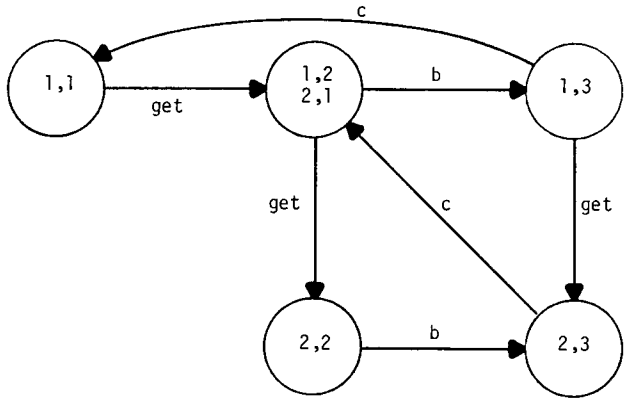


Fig. 7. Transition diagram obtained from Figure 5 by projection, ignoring the X-transition considered irrelevant.

where "−" means set subtraction; that is, $(V_0 - V_1)$ is the set of all actions that are in V_0 but not in V_1.

As far as the possible operation sequences are concerned, it is shown in the following section that a specification for an additional module M_2, if it exists, is given by the formula

$$S_2 = p_2(S_0 \times S_1) - p_2(\bar{S}_0 \times S_1) \tag{1}$$

where $\bar{S}_0$ is the complement of S_0, that is, the set of all execution sequences over operations in V_0, but in an order that is not possible according to the specification S_0.

We demonstrate this formula by the example considered above. We have seen that a *BUFFER* module could be implemented by two 3-*CYCLE* modules. Let us assume that the specification S_0 for our system M_0 is the *BUFFER* specification of Figure 2 and that we have decided that one of the submodules M_1 is a 3-*CYCLE* module executing the operations $V_1 = \{PUT, B, C\}$, with the possible execution sequences defined by $(PUT\ B\ C)^*$. Following the considerations above, we find that M_2 should execute the operation *GET* and should interact with M_1 through direct coupling with the actions B and C. Therefore, the operations of M_2 should be $V_2 = \{GET, B, C\}$. The possible execution sequences may be obtained from the formula above in several steps, as discussed below.

Formula (1) states that the possible execution sequences of M_2 are those sequences s_2 of actions in V_2 that satisfy the following:

(1) s_2 is the projection onto V_2 of some sequence in $S_0 \times S_1$, and
(2) there is no sequence s' in $\bar{S}_0 \times S_1$ such that its projection onto V_2 is equal to s_2. This implies that all possible execution sequences of the interacting submodules M_1 and M_2 satisfy S_0 (see Section 4).

We note that Figure 5 (assuming that the symbol X stands for *PUT*) shows the possible sequences of $S_0 \times S_1$. A transition diagram for the projection onto V_2 is obtained by replacing the *PUT* transitions by spontaneous transitions, since *PUT* is not an element of V_2. Considering only the accepting states and noting that the states $\langle 1, 2\rangle$ and $\langle 2, 1\rangle$ (which are connected through a spontaneous transition) may be combined, we obtain the diagram of Figure 7. Therefore, all execution

sequences that are possible according to this diagram satisfy the first condition above.

Not all of these sequences, however, satisfy the second condition. The second condition may be checked using Figure 5, noting that it is a transition diagram for $\bar{S}_0 \times S_1$ provided the accepting and nonaccepting states are interchanged. (I.e., the states $\langle 3, 1\rangle$, $\langle 3, 2\rangle$, and $\langle 3, 3\rangle$ are accepting; X stands for PUT, as before. We note that this simple transformation is valid if the diagram is deterministic. If it is nondeterministic, an equivalent deterministic diagram must first be found [9].) Now we can check the second condition for all the execution sequences that are possible according to Figure 7:

(1) The GET transition leading to the state $\langle 2, 2\rangle$ is not allowed, since its execution would be possible after an initial sequence of the form

$$(B\ (GET\ C + C\ GET))^*.$$

Therefore, the whole sequence is a projection of a sequence of the form

$$(PUT\ B\ (GET\ C + C\ GET))^*$$

that may lead to the state $\langle 3, 1\rangle$ in Figure 5.

(2) The C transition leading to the state $\langle 1, 1\rangle$ is not allowed, since such an execution sequence is the projection of a sequence that may lead to the state $\langle 3, 2\rangle$ in Figure 5 (through the spontaneous transition from $\langle 1, 1\rangle$).

The only remaining sequences are cyclic executions of the form

$$(B\ GET\ C)^*.$$

It is easy to check that these sequences satisfy the second condition. We have therefore shown that the module M_2 should be a 3-$CYCLE$ module participating in the actions B, GET, and C in a cyclic order.

It is shown in the next section that the specification S_2 obtained by formula (1) is a maximal one (including the largest number of execution sequences possible, corresponding to the most general module) satisfying

$$p_0(S_1 \times S_2) \subseteq S_0 \tag{2}$$

and

$$\text{each sequence } s_2 \text{ in } S_2 \text{ is compatible with } S_1. \tag{3}$$

We say that s_2 is compatible with S_1 if and only if there is some sequence s such that $p_2(s) = s_2$ and $p_1(s) \subseteq S_1$.

There are two important points to be noted here:

(1) The formula yields the most general module M_2 such that all execution sequences of M_2 may actually occur during interaction with the submodule M_1 (this follows from condition (3)), and

(2) it is not guaranteed that all execution sequences specified for M_0 will be obtained by the two interacting modules M_1 and M_2 (see condition (2)). Therefore, the construction of the submodule specification discussed in this section is at the same time stronger and weaker than the approach of verifying a possible implementation, discussed in Section 2.3.

Let us explain the difference considering the *BUFFER* implementation example. The implementation verification checks that a given implementation consisting of two 3-*CYCLE* submodules interacting in an appropriate manner satisfies the specification of the *BUFFER* modules. On the other hand, the submodule construction approach shows that, given a partial implementation of a *BUFFER* module by a 3-*CYCLE* submodule M_1, the remaining submodule should also be a 3-*CYCLE* submodule with a particular coupling, and that this is the most general submodule possible for this purpose. It does not show that the submodule M_2 so obtained, together with the given submodule M_1, will realize all execution sequences specified for the *BUFFER* system; however, this may be shown by verifying the obtained implementation.

Together, submodule construction and implementation verification allow one to determine whether, with given specifications for the module M_0 and one submodule M_1, there is a possible submodule M_2 (or any group of submodules) such that all these submodules will implement a system satisfying the specification for M_0. This is not possible by any other method (to the knowledge of the author). One would apply the submodule construction first and then verify the obtained implementation. If it is not a full implementation (i.e., does not realize all execution sequences specified for M_0), then there is none with the given module M_1.

There may be two reasons why not all execution sequences of M_0 can be implemented with a given M_1 and some suitable M_2. One reason could be that some execution sequences of S_0 are incompatible (see above) with the specification of M_1. In the case of the *BUFFER*, for example, the specification for M_1 given in Figure 8a would be incompatible with the specification for M_0 given in Figure 2. Another possible reason is more subtle and involves the interaction between the two submodules. An example is shown in Figures 8b through 8d. The specification of M_1 given in Figure 8b seems not to be in contradiction with the specification of Figure 2. However, formula (1) leads to the following conclusion. Figures 8c and 8d show transition diagrams for $S_0 \times S_1$ and $p_2(S_0 \times S_1)$, respectively. Checking the second condition (as in the example above), it is easy to see that the only possible execution sequence of M_2 is a single B transition. Therefore, a *GET* action will never be executed, and, clearly, most *BUFFER* execution sequences will not be implemented.

As shown by the examples of Section 5, the specification of M_2 obtained by the construction approach may be "too general," in the sense that it includes execution sequences that reduce the efficiency of the implementation. The implementation verification may also be used to check which execution sequences (if any) may be eliminated from the specification of the submodule M_2, and whether the obtained implementation is live and, in particular, has no deadlocks.

4. FORMAL DISCUSSION OF THE CONSTRUCTION OF SUBMODULE SPECIFICATIONS

We assume in this section the model of interacting modules discussed in Section 2. We consider a module M_0 consisting of n submodules M_i ($i = 1, 2, \ldots, n$). We assume that V is the set of all atomic actions that may occur in the system. An interaction occurs when several submodules participate in the execution of one

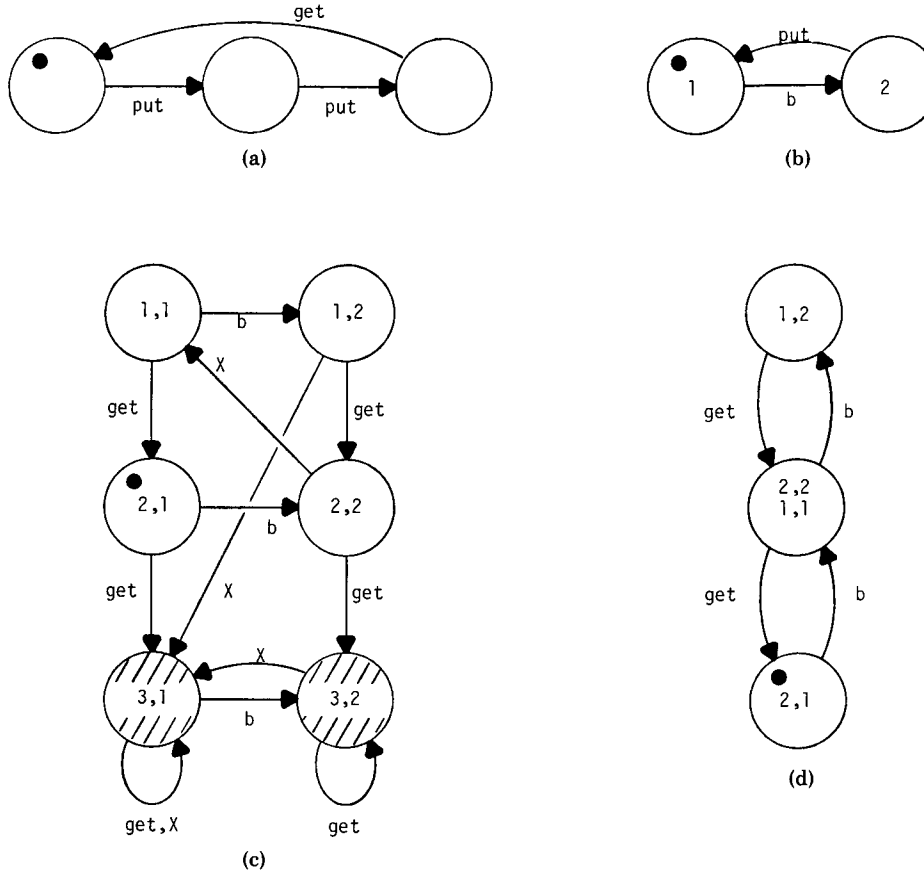


Fig. 8. Invalid implementation of a *BUFFER*. (a) Transition diagram of a module not compatible with the *BUFFER* specification of Figure 2. (b) Possible specification for a submodule M_1. (c) Transition diagram for $S_0 \times S_1$, where S_0 and S_1 are defined in Figures 2 and 8b, respectively. (d) Transition diagram obtained from Figure 8c by projection, ignoring the X-transition considered irrelevant.

atomic action. As above, we write V_i for the set of actions in which the module M_i participates, and $p_i(s)$ for the projection of an execution sequence s over V onto the set of (relevant) actions V_i.

Considering, for the moment, two submodules interacting with one another, but independent of any other part of the system, we may call this subsystem the "coupled product" of these submodules. It is defined by the sequences of atomic actions it realized, which are those sequences over $(V_i \cup V_j)$ that satisfy the specifications of both submodules. We write "$S_i \times S_j$" for this set of sequences and have

$$S_i \times S_j = \{s \in (V_i \cup V_j)^* \mid p_i(s) \subseteq S_i \wedge p_j(s) \subseteq S_j\}. \tag{4}$$

By definition, the operation of forming a coupled product is transitive. This corresponds to the fact that the order in which more than two submodules are

combined into a system has no influence on the behavior of the system, as long as the resulting coupling of submodules is the same.

We consider again the two approaches of implementation verification and construction of submodule specification. In the case of implementation verification, the traditional approach of program verification and consistency proofs for multilevel specifications, an implementation of a module M_0 (defined by V_0 and S_0) is given in terms of n interacting submodules M_i (defined by V_i and S_i, $i = 1, 2, \ldots, n$), and the condition

$$p_0(S_1 \times S_2 \times \cdots \times S_{n-1} \times S_n) = S_0 \tag{5}$$

and liveness are verified. Usually, this verification is done in two steps, proving, first, the so-called "partial correctness" or "safeness," which is expressed by the condition

$$p_0(S_1 \times S_2 \times \cdots \times S_{n-1} \times S_n) \subseteq S_0 \tag{6}$$

and, second, the liveness of the implementation. We assume that a "full implementation" of the specification S_0 is required, that is, that the condition

$$p_0(S_1 \times S_2 \times \cdots \times S_{n-1} \times S_n) \supseteq S_0 \tag{7}$$

(i.e., each sequence of S_0 can be realized by the implementation) is satisfied.

Condition (7) does not necessarily imply liveness. A counterexample is given by a *BUFFER* module implemented by two submodules M_1 and M_2 defined by

$$S_1 = (PUT\ (A\ B + C\ D))^*$$

and

$$S_2 = (A\ GET\ B + C\ GET)^*.$$

For this implementation, conditions (6) and (7) (and therefore condition (5)) are satisfied, but the execution of the atomic action C leads to a blocked system state where the only possible next operation is D for M_1, but A or C for M_2.

Statement about the Construction of Submodule Specifications. The approach of constructing a submodule specification using formula (1) solves the following problem. Given a module M_0 (defined by V_0 and S_0), submodules M_i (defined by V_i and S_i, $i = 1, 2, \ldots, n-1$), and the set V_n of atomic actions in which an additional submodule M_n participates, formula (1) defines a (maximal) specification S_n for M_n such that condition (6) is satisfied and for any sequence s_n over V_n that is not in S_n either

$$S_1 \times S_2 \times \cdots \times S_{n-1} \times \{s_n\} = \varnothing$$

(i.e., it is incompatible with the other submodules) or

$$p_0(S_1 \times S_2 \times \cdots \times S_{n-1} \times \{s_n\}) \not\subseteq S_0$$

(i.e., it does not satisfy the safeness condition).

This problem may be reduced to the problem of only one given submodule. We may define this submodule M_1' by

$$S_1' = p_1'(S_1 \times S_2 \times \cdots \times S_{n-1}) \tag{8}$$

where p_1' projects onto a set of relevant actions which should include at least those actions of $V_1 \cup V_2 \cup \cdots \cup V_n$ which contribute to S_0. We therefore consider in the following only the case of one given submodule M_1 and an additional submodule M_2 to be found.

PROPOSITION. *Suppose that $V_2 \subseteq (V_0 \cup V_1)$, that is, that each operation of M_2 is coupled either to M_0 or to M_1, or to both; then the sequences s_2 in $p_2(S_0 \times S_1)$ are exactly those sequences over V_2 for which*

$$p_0(S_1 \times \{s_2\}) \cap S_0 \neq \emptyset;$$

that is, they realize (together with some sequence $s_1 \in S_1$, and after projection onto V_0) a sequence of M_0.

PROOF. The following statements are equivalent:

(1) The sequence s_2 is in $p_2(S_0 \times S_1)$.
(2) There exists a sequence s over V such that

$$(s \in S_0 \times S_1) \wedge (p_2(s) = s_2).$$

(3) There exist sequences s over V and s_0 over V_0 such that

$$(p_0(s) = s_0) \wedge (p_2(s) = s_2) \wedge (p_1(s) \in S_1).$$

(4) There exist sequences s over V and s_0 over V_0 such that

$$(p_0(s) = s_0) \wedge (s \in (S_1 \times \{s_2\})).$$

(5) There exists a sequence s_0 over V_0 such that

$$s_0 \in p_0(S_1 \times \{s_2\}).$$ □

COROLLARY. *The sequences s_2 in $p_2(\bar{S}_0 \times S_1)$ are exactly those sequences over V_2 for which $p_0(S_1 \times \{s_2\}) \cap \bar{S}_0 \neq \emptyset$; that is, they realize (together with some sequences $s_1 \in S_1$, and after projection onto V_0) a sequence that is not in S_0.*

The above statement about the construction of submodule specifications using formula (1) follows directly from this proposition and its corollary.

To make formula (1) usable, one needs some specification language for defining the sets of possible execution sequences, such as S_1 or S_0. In this paper we only use FS-oriented specification. However, other specification languages (based on event counts or predicates, for instance) could be used in the submodule construction approach.

5. APPLICATION TO PROTOCOL DESIGN

We show in this section how the constructive approach discussed in the sections above may be applied to the design of protocols. In the case of two communicating protocol submodules, the module structure of the system is shown in Figure 1b. We assume that the service to be provided by the protocol is given (this is the specification S_0 of the system), and also the service of the underlying system layer, or "medium" (we call this specification S_3). If we assume that a specification of one of the communicating protocol submodules is given (we call this specification S_4), then we have the situation where $(n - 1)$ of n submodules (namely,

M_3 and M_4; and $n = 3$) are given for the implementation of a system, and the nth submodule (namely, M_2) is to be found. As discussed in Section 4, we may apply formula (1) for this purpose, where S_1 is defined by

$$S_1 = p_1(S_3 \times S_4). \tag{9}$$

5.1 The "Alternating-Bit" Protocol

As an example we consider the "alternating-bit" protocol (see, for example, [3]), which provides data transmission in one direction over a half-duplex transmission medium and recovers from detected transmission errors by retransmission.

Figure 9 shows the operations of the different submodules. Through the operations *PUT* or *GET* the modules using the communication system (not shown in the figure) submit a data block or retrieve it, respectively. At any given time, there is at most one data block in transit. The operations $\underline{d}_i$ and $\bar{d}_i$ ($i = 0$, 1) represent the sending and receiving, respectively, of an information frame which contains the last data block submitted by the user and the "alternating bit." Similarly, the operations $\underline{a}_i$ and $\bar{a}_i$ are the sending and receiving of an acknowledge frame, which contains only a single bit. The operations $\bar{d}_e$ and $\bar{a}_e$ are a reception of a frame in error (full error detection is assumed to be performed by the "medium" submodule; in practice, an error-detecting code would be used).

Figure 10a shows a diagram specifying the "medium" submodule. It defines a half-duplex medium with only a single frame in transit. Figure 10b specifies the "sender" protocol submodule. It shows the value of the "alternating bit," which changes its value for each new data block submitted by the user, and the retransmission in the case that the right acknowledgment is not received.

To find the specification S_2 of the "receiver" submodule we proceed as follows: First, we determine S_1 according to formula (9) using $V_1 = \{PUT, \underline{a}_0, \bar{a}_1, d_0, d_1, \bar{d}_e\}$. This leads to the diagram of Figure 10c. We note that this diagram is nondeterministic, reflecting the fact that a transmission error may or may not have been detected at the interface between the "sender" and "medium" submodules (the operations at this interface are not directly visible at this level of detail, because they have been lost during the projection p_1 of formula (9)).

For S_0 we adopt the specification shown in Figure 10d. This service specification (also discussed in [4]) is similar to the specification of the *BUFFER* (see Figure 2), but it includes an additional d_0, d_1 transition that ensures that a data frame is correctly received before a data block is retrieved by the user. (We note that the actions d_0 and d_1 have the participation of all modules considered here: M_1, M_2, and M_0). Now using formula (1), we obtain for the "receiver" M_2 the specification shown in Figure 10e. We note that this transition diagram includes the execution sequences of the traditional alternating-bit receiver [3] shown in Figure 10f. In addition to the latter, it allows for the sending of a negative acknowledgment after the correct reception of a data block (see fat transitions in Figure 10e). The resulting additional loops can only decrease the efficiency of the protocol and are undesirable.

This example demonstrates the fact that the specification obtained by our formula may be more general than required. As pointed out above, it finds the most general specification possible. (General specifications of the alternating-bit protocol are also considered in [15].) In order to find a more specific specification, more appropriate for our purposes, we may delete some transitions from the

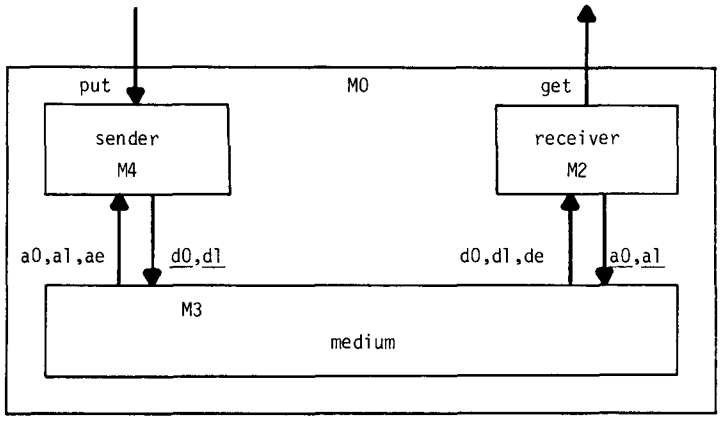


Fig. 9. The structure of a communication subsystem involving the alternating-bit protocol. The interactions between the submodules are indicated in the following notation:

PUT submitting a data block;
GET retrieving a data block;
$\underline{d_0}$, $\underline{d_1}$ sending an information frame;
$\overline{d_0}$, $\overline{d_1}$ receiving an information frame;
d_e receiving an information frame in error;
$\underline{a_0}$, $\underline{a_1}$ sending an acknowledgment frame;
$\overline{a_0}$, $\overline{a_1}$ receiving an acknowledgment frame;
a_e receiving an acknowledgment frame in error.

transition diagram of Figure 10e provided that the resulting FS machine, when executed together with a submodule satisfying S_1, will still provide all execution sequences specified by S_0, which may be checked by an implementation verification. In fact, the fat transitions in Figure 10e may be deleted, thus leading, after simplification, to the transition diagram of Figure 10f.

5.2 A Protocol Without Sequence Numbering

A similar protocol without sequence numbering has been considered in the literature. The "sender" submodule for this protocol is shown in Figure 11. (We note that there is no alternating bit, and "$a+$" and "$a-$" stand for positive and negative acknowledgment, respectively.) Doing a construction similar to that for the "alternating-bit" protocol above, one obtains a specification S_2 for the "receiver" submodule that allows only a finite number of transitions. Therefore, no submodule M_2 exists that, together with the "sender" submodule and the underlying medium, will provide the required communication service shown in Figure 10d. We note that this is a stronger result than showing (as done in [4]) that the sender submodule of Figure 10b does not operate correctly with a particular "receiver." The conclusion is reasonable, since without sequence numbering the "receiver" submodule will never be able to distinguish between a new data block and a retransmission.

5.3 Module Specifications with Parameterized Operations

In the service specification of Figure 10d, the transitions d_0 and d_1 were included to ensure that the data block retrieved by the user is the same as the one

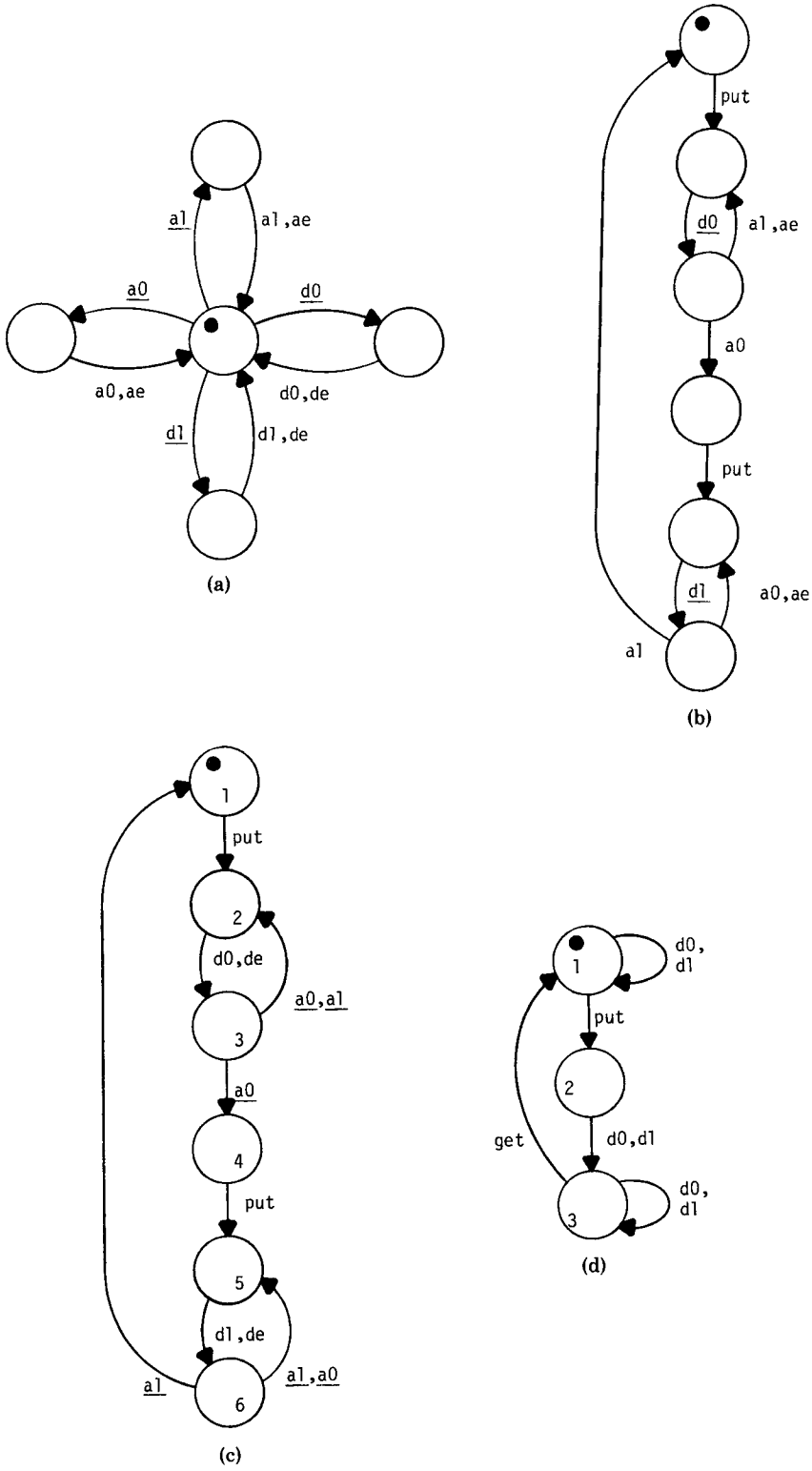


Fig. 10. The alternating-bit protocol. (a) Specification of the underlying "medium." (b) Specification of the "sender" submodule. (c) "Sender" and "medium" combined. (d) Specification S_0 of the service to be provided. (e) The resulting specification for the "receiver" submodule. (f) The traditional "receiver" submodule.

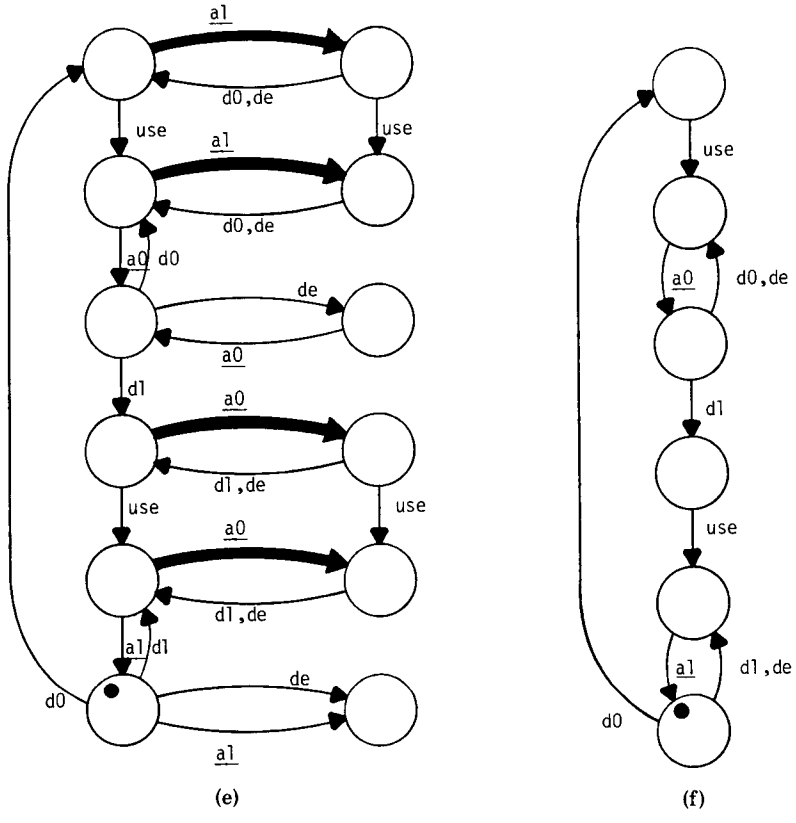


Figure 10 continued.

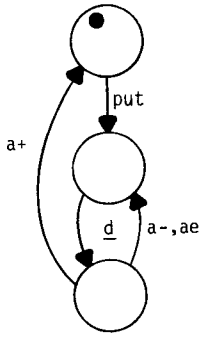


Fig. 11. Specification of the "sender" submodule for a protocol without sequence numbering.

submitted at the sending side. A more natural approach for ensuring this property is the use of parameterized operations. This approach leads us beyond the FS model used for the examples above. In this section, we consider the construction of the alternating-bit "receiver" submodule in such an extended formalism.

The service operations for the protocol submodules are "*PUT*(*x*)" and "*GET*(*x*)," where *x* is the value of a data block. We therefore consider as many different "*PUT*" operations as there are possible different data blocks, and

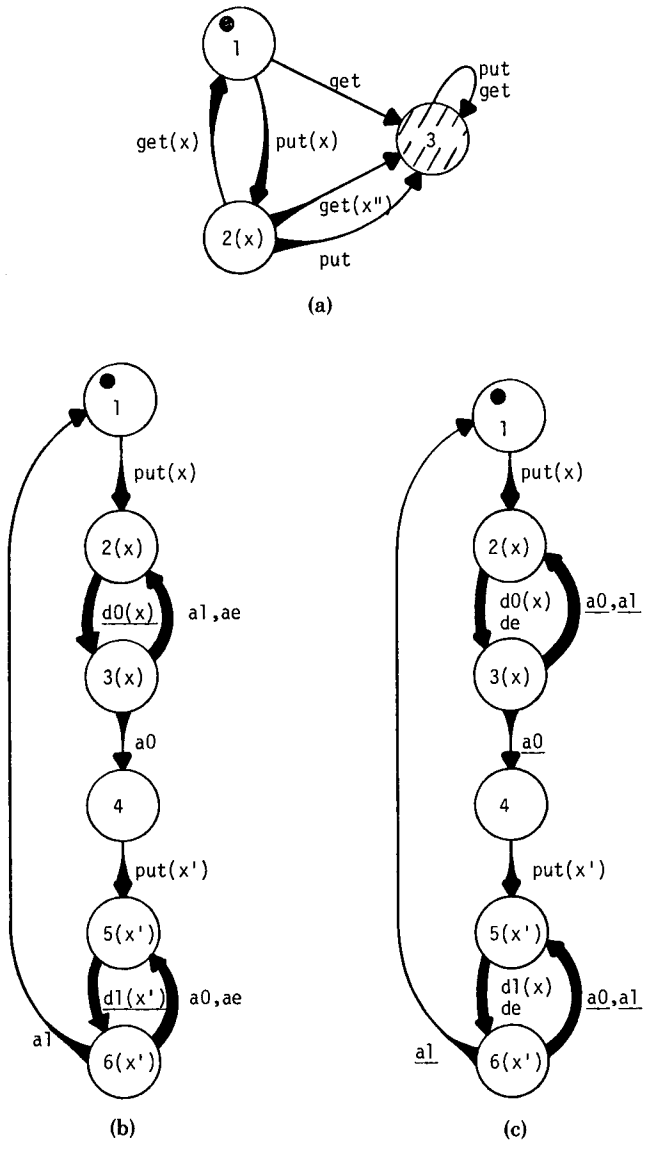
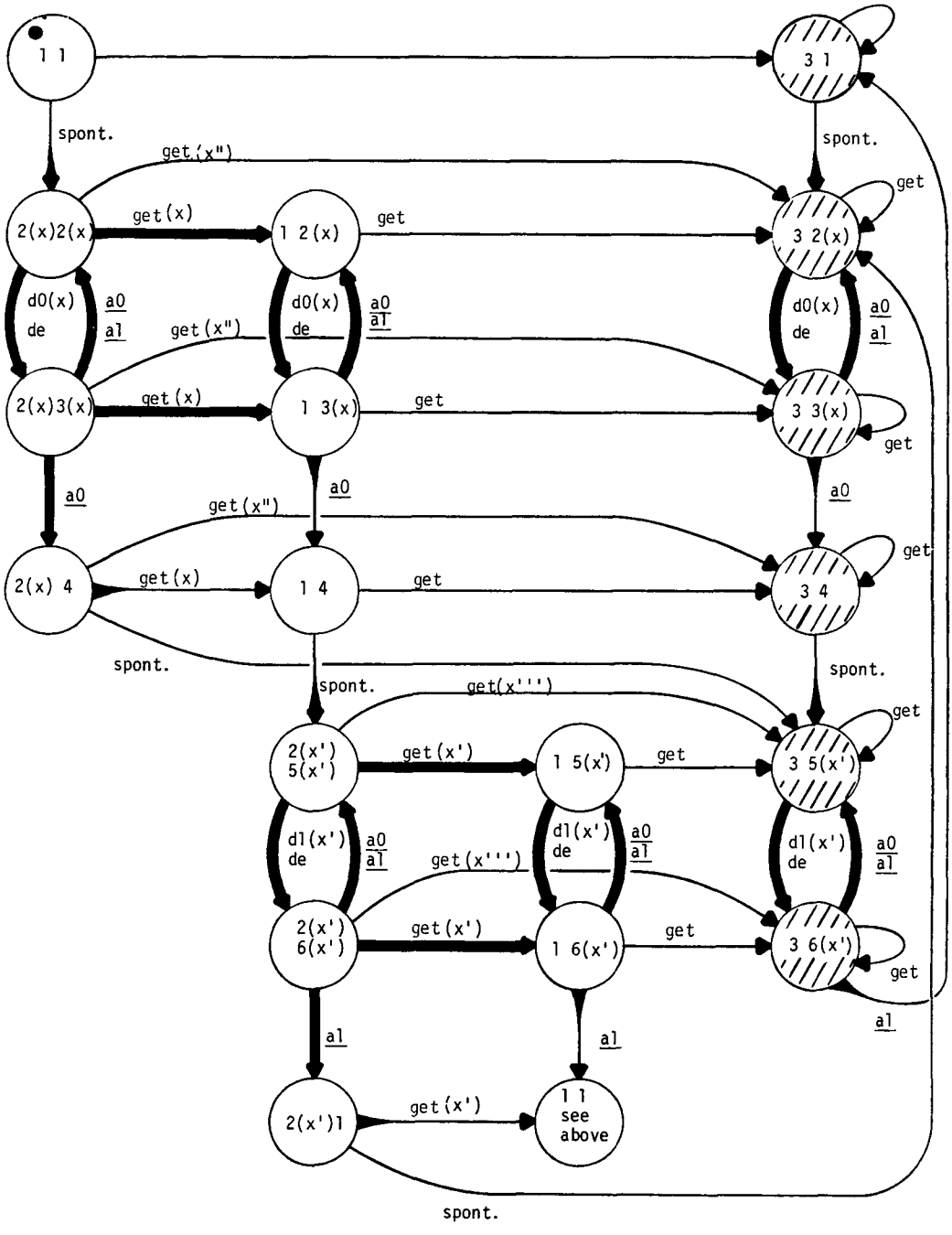


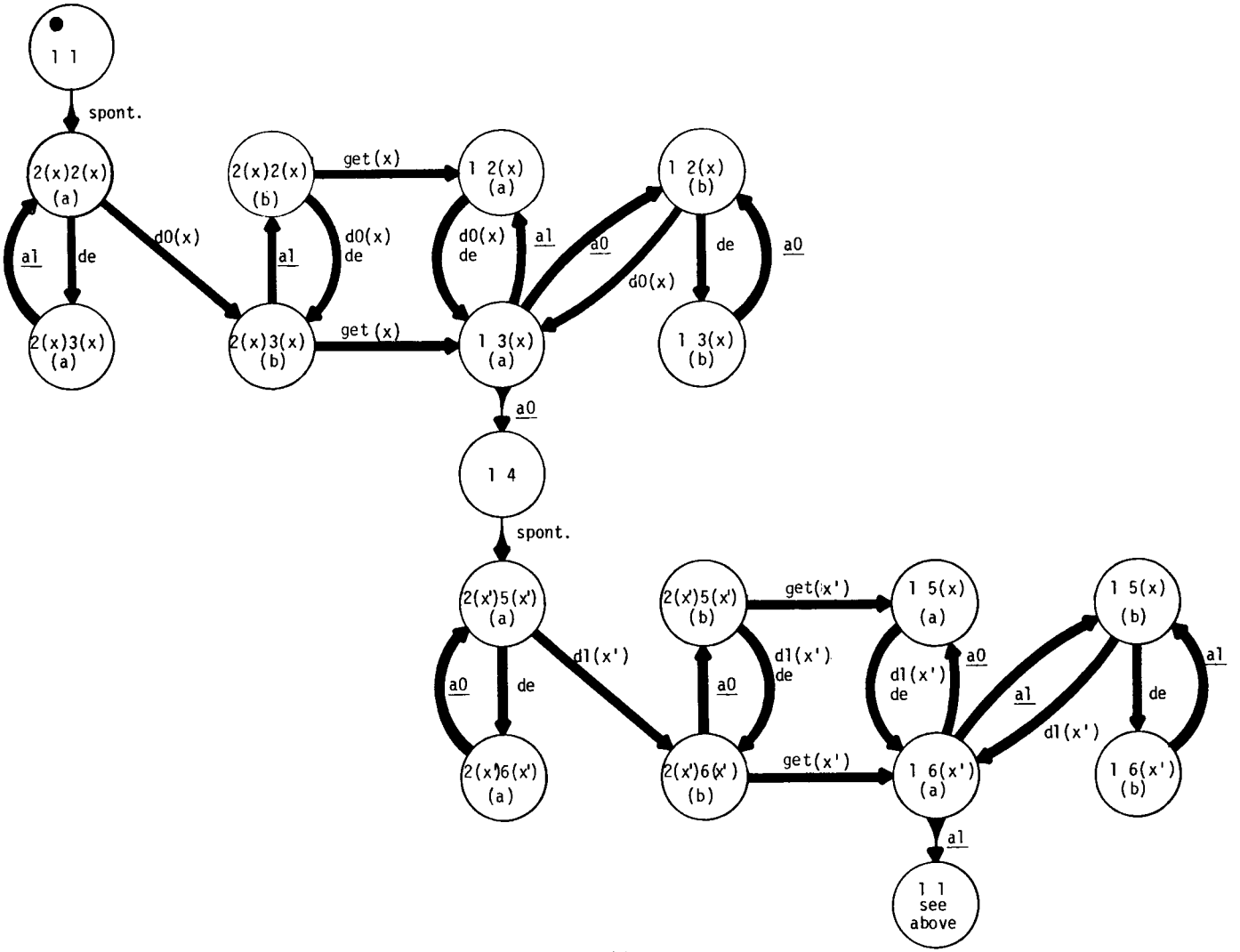
Fig. 12. The "alternating-bit" protocol specified with parameterized events. (a) Specification S_0 of service to be provided. (Note: $x'' \neq x$; "*PUT*" and "*GET*" stand for events with any arbitrary parameter values.) (b) Specification of the "sender" submodule. (c) "Sender" and "medium" combined. (d) $p_2(S_0 \times S_1)$. (Note: $x''' \neq x'$.) (e) Resulting specification for the "receiver" submodule. (f) Simplified "receiver" submodule.

similarly for "*GET*." The service specification S_0 is shown in Figure 12a, where the states $\langle 1 \rangle$ and $\langle 3 \rangle$ are single states, whereas $\langle 2(x) \rangle$ is a parameterized state which stands for a number of different states that are distinguished by different data block values. Here a fat end of a transition arrow indicates that the parameter values of the transition must correspond to the parameter value of the



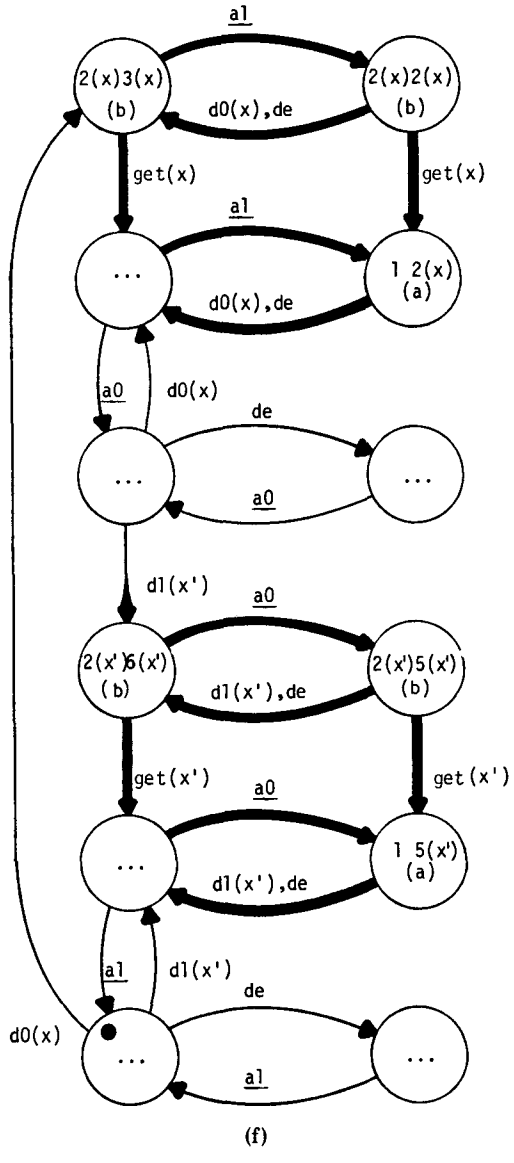
(d)

Figure 12 continued.



(e)

Figure 12 continued.



(f)

Figure 12 continued.

state. For example, the transition arrow "*PUT*(*x*)" stands for a number of different transitions from state ⟨1⟩ to different states ⟨2(*x*)⟩, one for each possible data block value. This service specification ensures that the data block of a "*GET*" operation is equal to the data block of the preceding "*GET*" operation.

Including the data block parameter in the "sender" submodule of Figure 10b yields the "sender" shown in Figure 12b. The "medium" of Figure 10a may also be easily extended to include data block parameters. Figure 12c shows the specification S_1 for the "sender" and the "medium," combined. We may now use formula (1) to find a possible "receiver" submodule. The first step is the construc-

tion of $p_2(S_0 \times S_1)$, which is shown in Figure 12d. By interchanging accepting and nonaccepting states, this figure also defines $p_2(\bar{S}_0 \times S_1)$, which specifies execution sequences not allowed for the "receiver" submodule. We may obtain a state diagram for the "receiver" from Figure 12d by the following considerations:

(1) Because of the spontaneous transition from state $\langle 2(x), 4\rangle$ to state $\langle 3, 5(x''')\rangle$, no "$a_0$" transition should be allowed after a sequence leading to the state $\langle 2(x), 4\rangle$. We therefore eliminate both "a_0" transitions from state $\langle 2(x), 3(x)\rangle$, and, similarly, the "$a_1$" transitions from state $\langle 5(x''), 3(x'')\rangle$.

(2) An initial "$GET(x)$" transition is not allowed, since it may lead to the state $\langle 3, 2(x)\rangle$ through a state $\langle 2(x'), 2(x')\rangle$ with a different x' and a transition labeled "$GET(x')$."

(3) Sequences such as "$d_e\ GET(x)$" are not allowed, since they may lead to the state $\langle 3, 3(x)\rangle$ through a path similar to the one considered above.

(4) After a sequence leading to the state $\langle 1, 3(x)\rangle$, a sequence such as "$a_0\ d_e\ a_1$" is not allowed, since it may lead to the state $\langle 3, 2(x)\rangle$ through the states $\langle 1, 4\rangle$, $\langle 2(x''), 5(x'')\rangle$, $\langle 2(x''), 3(x'')\rangle$, and $\langle 2(x''), 1\rangle$.

Taking into account these constraints, we may rearrange the transition diagram of Figure 12d into the "receiver" diagram shown in Figure 12e, where that part of the diagram not leading to accepting states has not been shown. It is easy to see that this diagram is equivalent to the diagram of Figure 12f. The latter is similar to Figure 10e, but, in addition, it includes appropriate constraints for the data block parameters.

6. CONCLUSIONS

This paper presents a method for constructing submodule specifications that may be used in the stepwise refinement of system specifications. The formula underlying the method is very general. It provides the specification for an additional submodule that, together with some already specified submodules, will provide the specified service (or a subset of the execution sequences defined by the service specification, if the complete service cannot be obtained with the submodules already specified).

The formula defines the most general submodule possible (allowing for the largest number of different execution sequences). This may not always be desirable. For instance, this leads to undesirable additional loops in the examples of Figures 10 and 12. It would be interesting to find a formula that yields minimal specifications for the additional submodule.

The problem of avoiding deadlocks and loops without progress has not been addressed in this paper. Therefore, this constructive method should be complemented with a more traditional verification method that checks that the implementation resulting from this submodule construction is a full implementation of the specified service and is live and efficient (and, in particular, contains no deadlocks).

Different specification languages may be used with the submodule construction method. The use of an FS-oriented language has been demonstrated in this paper. As an extension to the finite-state approach, an example involving parameterized operations is discussed in the section above.

In the case of an FS-oriented specification language, the number of states to be considered during the submodule construction process may be very large. Therefore, it would be interesting to automate the method for FS specification. A system for the interactive design of modular systems (for example, communication protocols) may be based on this method. In addition to finding the specification of an additional submodule, when this is possible, the system may also give useful information to the designer in those cases when no solution exists.

ACKNOWLEDGMENTS

We thank E. Cerny, J. Gecsei, S. Owicki, D. Stevenson, C. Sunshine, and the referees for many helpful comments on earlier versions of this paper.

REFERENCES

1. Bartussek, W., and Parnas, D.L. Using traces to write abstract specifications for software modules. Rep. TR 77-012, University of North Carolina, 1977.
2. Bochmann, G.V. Distributed synchronization and regularity. *Comput. Networks 3* (1979), 36–43.
3. Bochmann, G.V. Finite state description of communication protocols. *Comput. Networks 2*, 4/5 (Oct. 1978), 361–372.
4. Bochmann, G.V. Communication protocols and error recovery procedures. In Proceedings, ACM Interprocess Communication Workshop. *Oper. Syst. Rev. 9*, 3 (Mar. 1975), 45–50.
5. Campbell, R.H., and Habermann, A.N. The specification of process synchronization by path expressions. In *Lecture Notes in Computer Science*, vol. 16: *Operating Systems*. Springer-Verlag, New York, 1974.
6. Cerny, E. Controllability and fault observability in modular combinational circuits. *IEEE Trans. Comput. C-27*, 10 (Oct. 1978), 896–903.
7. Cerny, E., and Marin, M.A. An approach to unified methodology of combinational switching circuits. *IEEE Trans. Comput. C-26*, 8 (Aug. 1977), 745–756.
8. Hoare, C.A.R. Communicating sequential processes. *Commun. ACM 21*, 8 (Aug. 1978), 666–677.
9. Hopcroft, J.E., and Ullman, J.D. Formal languages and their relation to automata. Addison-Wesley, Reading, Mass., 1969.
10. LeGuernic, P., and Raynal, M. Eléments d'un langage adapté à la communication entre processus. In Actes du Congrès AFCET (Association Française pour la Cybernetique, Economique et Technique), Nancy, France, Nov. 1980, pp. 667–676.
11. Luckham, D.C., and Karp, R.A. An axiomatic semantics of concurrent cyclic processes. Tech. Rep., Artificial Intelligence Laboratory, Stanford University, Stanford, Calif., 1979.
12. Milne, G., and Milner, R. Concurrent processes and their syntax. *J. ACM 26*, 2 (Apr. 1979), 302–321.
13. Milner, R. Four combinators for concurrency. In ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada, Aug. 18–20, 1982, pp. 104–110.
14. Prinoth, R. Eigenschaften faerbbarber Petri-Netze. Tech. Rep. TRG, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, W. Germany, 1977.
15. Schwartz, R.L., and Melliar-Smith, P.M. Temporal logic specification of distributed systems. In Proceedings, 2nd International Conference on Distributed Systems, Paris, Apr. 8–10, 1981.
16. Zafiropulo, P., West, C.H., Rudin, H., Cowan, D.D., and Brand, D. Towards analysing and synthesizing protocols. *IEEE Trans. Commun. Com-28*, 4 (Apr. 1980), 651–661.

Received November 1979; revised June 1980 and November 1981; accepted November 1981