Modular Composition of Redundancy Management Protocols in Distributed Systems: An Outlook on Simplifying Protocol Level Formal Specification & Verification[*]

Purnendu Sinha
ECE Dept.
Concordia University
Montréal, Canada
sinha@ece.concordia.ca

Neeraj Suri
Dept. of Computer Engineering
Chalmers University
Göteborg, Sweden
suri@ce.chalmers.se

Abstract

In recent years, formal methods (FMs) have been extensively used for verification and validation (V&V) of dependable distributed protocols. Over our studies in utilizing FMs for V&V, we have observed that a number of protocols providing for distributed and dependable services can often be formulated using a small set of basic functional primitives or their variations. Thus, from the formal viewpoint, the objective of this paper is to introduce techniques, utilizing concepts of category theory, that could effectively identify and reuse basic formal modules in order to simplify formal specification and verification for a spectrum of protocols.

1 Introduction

In a dependable distributed environment, redundancy in the physical or temporal domain is a well established approach for providing reliable services. For dependable systems, the ability to correctly detect, diagnose, and recover from errors becomes a significant design consideration for protocols handling each of these specific tasks. Typically, the design and testing of dependable distributed protocols involves investigating the large number of state space and possible execution paths in the protocol operations. The conventional techniques, utilizing simulation and prototyping over selected sample cases, are severely limited in handling complex protocol operations. As formal methods [3] provide an extensive support for automated and exhaustive state explorations over the formal verification based analysis of operations of a given protocol, in [14, 13] we introduced a formal methods based approach to specifically identify pertinent test cases to guide the experimental validation process. Over these studies, we observed that most protocols that provide for dependable services required us to model just a few basic functional primitives for its testing. Thus, our objective from the formal viewpoint is to investigate whether these functional primitives can be formally characterized for their subsequent reuse in specifying and verifying generic distributed dependable protocols. For example, formalizing a complex protocol could be made easier if guidelines could be provided as per the reuse of the formal specification and verification of individual functional building blocks of the overall protocol. Hence, to support our philosophy of "reuse of primitives", our specific objectives here are to develop a modular approach for protocol composition through:

- Identifying building blocks within the class of redundancy management protocols,
- Highlighting and addressing issues involved in block interactions and inter-dependencies.
- Utilizing concepts of category theory to develop guidelines for protocol composition and construction of libraries of formal specifications (and associated verification) of categorized building-block protocols.

We begin with highlighting the distinction of our approach towards modularization from other "building block" approaches.

[*]Supported in part by TFR, NSF CAREER CCR 9896321 (Suri) & FRDP Grant of Concordia Univ. (Sinha)

Related Work: Modularization is a well-known technique for simplifying complex software systems [1, 2, 5, 6, 8, 10, 15]. We have observed that most of the approaches [5, 6, 10, 15] focus on *implementation* aspects of the composition of a protocol from micro-protocols developed for specific services. Among other approaches [1, 2, 8] have provisions for formal reasoning to ascertain configurations against system specifications. Our approach utilizes category theory based concepts to define interfaces for basic modules which a protocol designer can use and be aware of at the time of selecting modules to configure complex protocol-level operations. We envision that the proof of correctness of the composite protocol can be established utilizing the *proof constructs* being developed for these basic building blocks.

Organization: Section 2 introduces a category theory based approach for modular composition, and presents our building block approach to formal specification and V&V of dependable protocols. Sections 3.1 through 3.5 highlight attributes of constituent building blocks of redundancy management protocols Utilizing these building blocks, Section 3.6 presents hierarchical composition of a fault detection, isolation and reconfiguration (FDIR) protocol to illustrate our modular approach. We conclude with a discussion in Section 4.

2 The Building Block Approach to Protocol Composition and V&V

In this section, we present a formal framework which utilizes concepts of category theory to facilitate a rigorous and consistent composition out of system building-block protocols. Figure 1 depicts our general approach for formal-method-based V&V of dependable distributed protocols.

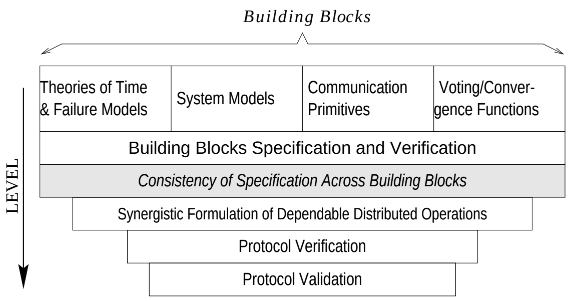

Building Blocks
Theories of Time & Failure Models
System Models
Communication Primitives
Voting/Convergence Functions
Building Blocks Specification and Verification
Consistency of Specification Across Building Blocks
Synergistic Formulation of Dependable Distributed Operations
Protocol Verification
Protocol Validation
LEVEL

Figure 1: A General Framework

We note that *agreement, synchronization*, and *detection* are essential functional primitives for developing most dependable applications. Over our studies, we have observed that the following four primary protocol building blocks, namely: **(a)** system models, **(b)** failure models, **(c)** communication primitives, and **(d)** voting/convergence functions are typically being used in developing most fault-tolerant services. We will discuss these in detail in Section 2.2. We next discuss applicability of category theory concepts to modular composition of spectrum of dependable protocols.

2.1 Category-Based Modular Composition

We have adapted *calculus of modules* based on categorical concepts [4, 9, 11] in simplifying formal verifications. We first define few general terms; for details, the reader is referred to [4, 11].

- **Signature:** A signature $SIG = (S, OP)$ consists of a set S, the set of sort, and a set OP, the set of constant and operation symbols.
- **Specification:** A specification $SPEC = (SIG, AX)$ consists of two parts: the signature SIG and a set of axioms AX which describes the behavior of the system as well as constraints on the environment.
- **Specification Morphism:** A specification morphism $m : SPEC1 \to SPEC2$ from a specification $SPEC1$ to a specification $SPEC2$ maps any element of the signature of $SPEC1$ to an element of the signature of $SPEC2$ that is compatible. Moreover, m must be such that image of any axiom of $SPEC1$ is a theorem of $SPEC2$.

In order to define module specifications, we utilize the notion of push-out operation from the category theory. Given specifications A and B, and a specification R describing syntactic and semantic requirements along with two morphisms f and g, the push-out operation gives specification P which contains A and B (See Figure 2).

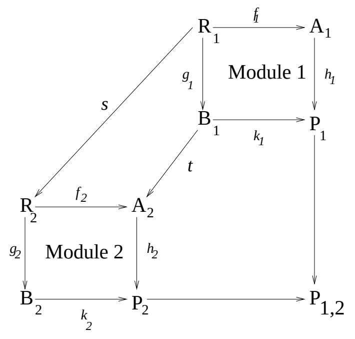
$$\begin{tikzcd}
R \arrow[r, "f"] \arrow[d, "g"'] & A \arrow[d, "h"] \arrow[ddr, "h'"] & \\
B \arrow[r, "k"] \arrow[drr, "k'"'] & P \arrow[dr, "u"] & \\
 & & P'
\end{tikzcd}$$

Figure 2: Push-out

Formally, given specification morphism $f : R \to A$ and $g : R \to B$, a specification P together with specification morphisms $h : A \to P$ and $k : B \to P$ is called **push-out** (of f and g), if we have $h \circ f = k \circ g$, where $\circ$ denotes composition. Furthermore, for all specification P' and morphisms h' and k' such that $h' \circ f = k' \circ g$, there exists an unique morphism $u : P \to P'$ such that $u \circ h = h'$ and $u \circ k = k'$. The specification P is the complete description of the module.

In general, protocols utilize services rendered by other protocols, and extend their services to be used in conjunction with other protocols to achieve the overall desired objective. In this respect, specifications A and B can constitute interfaces of the module. Specification B could declare attributes/operations that must be imported from other modules, and similarly, specification A could declare attributes/operations that can be exported to other modules. It is to be emphasized that interaction or relationship between the modules are expressed by means of morphisms, and categorical operations assist constructing larger modules resulting from these interactions.

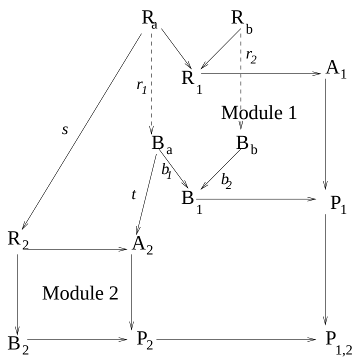
Composition: In order to capture the protocol or module interactions, we propose a *composition* scheme. Under this scheme two modules are interconnected via export and import interfaces. The push-out of two modules is the resulting specification of the composed module. Figure 3 depicts the composition operation. Here, *Module 1* imports via specification B_1 whatever *Module 2* exports via specification A_2. The compatibility of the parameters (or semantic constraints) is governed by the morphism s. Furthermore, the following property must be respected: $t \circ g_1 = f_2 \circ s$. In this case, the resulting module is $(R_1, B_2, A_1, P_{1,2})$, where $P_{1,2}$ is the push-out of P_1 and P_2 over B_1.



$$\begin{tikzcd}
 & R_1 \arrow[r, "f_1"] \arrow[d, "g_1"'] \arrow[ddl, "s"'] & A_1 \arrow[d, "h_1"] \\
 & B_1 \arrow[r, "k_1"'] \arrow[dl, "t"] & P_1 \arrow[dd] \\
R_2 \arrow[r, "f_2"] \arrow[d, "g_2"'] & A_2 \arrow[d, "h_2"] & \\
B_2 \arrow[r, "k_2"'] & P_2 \arrow[r] & P_{1,2}
\end{tikzcd}$$
Module 1
Module 2

Figure 3: Composition involving single import

For more complex protocols, a module may import parameters from several different modules, and also the specification consisting of syntactic and semantic requirements may compose of several different small specifications. Figure 4 depicts a general overview of the scheme. Here, the import specification B_1 of *Module 1* contains elements which B_a imports from *Module 2* and elements imported by B_b. If two different modules share some attributes, then a new module is constructed which contains attributes or actions common to the two modules, and module morphisms are defined to allow such associations. Essentially, the goal here is to decompose global properties into elementary lemmas provable in basic modules, and translate these lemmas along morphisms to obtain the desired properties. We will highlight nuances of this scheme when we discuss the modular composition of the FDIR protocol in Section 3.6.



$$\begin{tikzcd}
 & R_a \arrow[d, dashed, "r_1"'] \arrow[dr] \arrow[dddl, "s"'] & R_b \arrow[d, "r_2"] & \\
 & & R_1 \arrow[r] \arrow[d, dashed] & A_1 \arrow[dd] \\
 & B_a \arrow[dr, "b_1"] \arrow[dd, "t"'] & B_b \arrow[d, "b_2"] & \\
 & & B_1 \arrow[r] & P_1 \arrow[dd] \\
R_2 \arrow[r] \arrow[d] & A_2 \arrow[d] & & \\
B_2 \arrow[r] & P_2 \arrow[rr] & & P_{1,2}
\end{tikzcd}$$
Module 1
Module 2

Figure 4: Composition involving multiple imports

We emphasize that a categorical-based approach is presented to assist a protocol designer to choose constituent modules and test for their compatibility as part of the selection process. With this background, we now highlight the salient features of basic building blocks mentioned on the top most level in Figure 1.

2.2 Basic Building Blocks System Models

Within a typical model of distributed systems, the synchrony of the system relates to assumptions made about the time bounds on the performance of the system. We outline pertinent aspects of synchrony through characterizing its functions as a module given below. We will utilize this format of "module" throughout the paper.

Module Name: Synchronous System[1]
Module Attributes:

[1]Our approach directly extends to asynchronous, timed-asynchronous and quasi-asynchronous models as well.

- For a chosen message type, transmission and processing delays are bounded by a constant d; this consists of the time it takes for sending, transporting, and receiving a message over a link.
- Every process p has a local clock C_p with known bounded rate of drift $\rho \geq 0$ with respect to real-time. That is, for all p and all $t > t'$,

$$(1+\rho)^{-1} \leq \frac{C_p(t) - C_p(t')}{(t - t')} \leq (1+\rho)$$

  where $C_p(t)$ denotes p's local clock time at real-time t.
- The clocks of correct processors are monotone increasing functions of real-time and the resolution of processors clocks is fine enough, so that separate clock readings yield different values, i.e., $t_1 \leq t_2 \leftrightarrow C_p(t_1) \leq C_p(t_2)$.

Failure Models

A *failure model* specifies how a faulty component can deviate from its specification. For fault diagnosis, the processor fault classes considered in the literature are *transient, intermittent*, or *permanent*. These common fault classes characterize errors which occur in the data domain. Another classification of faults is based on the perturbations that occur (and are being detected) in the time domain. The classes, from strongest to weakest, are *fail-stop* faults, *crash* faults, *omission* faults, *timing* faults, and *Byzantine* faults.

Communication Primitives

Generally, communication protocols utilize a datagram service which allows the transmission of messages along links between a pair of nodes in a point-to-point communication network. The datagram service has omission or performance failure semantics.

Voting/Convergence Functions

Redundancy implies having multiple system entities execute the same task, and compare their outputs. The primary redundancy technique employed for detecting errors has been *duplication*. If at least three processors are involved (e.g., TMR), the comparison can be performed through majority voting (e.g. 2-out-of-3). [See Section 3.1]

2.3 Basic Blocks $\rightarrow$ Protocol Specification

After having outlined the basic primitives or building blocks, the next step is to formally specify these basic primitives identifying their export and import interface, and their associated morphisms. Based on these primitives, we have to construct specifications for system building blocks inherently being used in redundancy management protocols. We stress the fact that explicit declaration of export and import interfaces of a module specification facilitate us to establish the consistency of the requirements across these modules to avoid any conflicting specifications.

2.4 Protocol Verification

As category theory allows composition of a larger module specification from small module specifications, we exploit this modular structure to simplify formal verification. For each component in module interactions, the goal is to define specification morphisms capturing different nuances. Also, there is a need to show that the image of any axiom from a source specification is a theorem in the goal specification along the morphism(s) connecting them. Hence, we decompose global properties into elementary lemmas provable in more basic specifications, and these lemmas are translated along morphisms to the global description.

2.5 Protocol Validation

For completeness, we briefly discuss protocol validation aspects of our building block approach. We utilize our developed representation structures (refer to [14, 13] for details) to encapsulate protocol attributes generated over the formal specification and verification process to identify system states and design/implementation parameters to construct test cases.

3 Redundancy Management Operations

Having thus far outlined our category theory based modular approach, we now focus on characterizing each functional building block used in the redundancy management procedures. Once a chosen system model and associated fault/error model is established, the empirical formulation of a generic redundancy management protocol involves the following procedures:

- Specifying the procedures for assimilating varied information obtained from the redundant system entities, and disseminating (and pruning) it to obtain a value(s) usable in subsequent protocol operations, i.e., voting/convergence etc.
- Characterizing the specific operational features, within the chosen system models, that are essentially required in formulating any redundancy management procedures. Most dependable protocols typically employ the following four functions (blocks): **(a)** synchro-

nization, **(b)** atomic broadcast, **(c)** agreement, and **(d)** checkpoint establishment.

For each of these blocks, we outline primary attributes; for details, we refer the reader to [7, 12].

3.1 Building Block 1: Voting/Convergence Functions

As the *voting/convergence function* is a primary building block for redundancy management operations, we highlight their properties of interest.

Module Name: Voting/Convergence Function
Module Attributes:

- All N redundant units perform identical operation using identical inputs. N must be at least $2m+1$ or $3m+1$ for NMR or Byzantine-resilient systems, respectively, where m is the maximum number of faulty units.
- All inputs to the voter must be from the same cycle. The order of the input sequence must be sustained.
- Voter processing time must be less than the incoming message rate.

3.2 Building Block 2: Synchronization

Redundancy requires some form of synchronization among different processors. Each processor periodically executes a protocol that involves exchanging clock values with the other processors, computing a reference value, and appropriately adjusting its local clock to reflect the consensus.

Module Name: Synchronization
Module Attributes

- The non-faulty clocks are initially synchronized to some constant quantity.
- The non-faulty clock should not drift away from real-time by a rate greater that ρ.
- The period between two re-synchronization signals and the range within which these signals occur must satisfy defined bounds.
- There is no overlap between synchronization periods.
- A non-faulty processor can read the difference between its own clock and that of another non-faulty processor with at most a small error.

3.3 Building Block 3: Atomic Broadcast

An important objective in fault-tolerant distributed system is to provide consistent information to multiple processors in the system. Broadcast services are useful in many applications such as managing process groups, commit protocols, etc.

Module Name: Atomic Broadcast
Module Attributes:

- The network remains connected even upon failures in components (processors and links).
- The clocks of correct processors are approximately synchronized within a maximum allowable deviation.
- Transmission and processing delays of messages are bounded by a constant.

3.4 Building Block 4: Exact Agreement

Distributed system functions often require all non-faulty processors to agree mutually on a specific value, even if certain processors in the system are faulty. This agreement is achieved through an agreement protocol that involves several rounds of message exchange among the processors.

Module Name: Exact (Byzantine) Agreement
Module Attributes:

- At least $3m+1$ participants must be there. m is the number of faulty processors.
- Each participants must be connected to each other participants through at least $2m+1$ disjoint communication paths.
- At least $m+1$ rounds of communication among participants must take place.
- All participants must be synchronized within a known skew of each other.

3.5 Building Block 5: Checkpointing

Checkpointing is a commonly used technique to provide for sustained operations in a distributed system in the presence of transient faults, without incurring the high performance cost of restarting tasks/processes from scratch as transients are encountered.

Module Name: Checkpointing
Module Attributes

- The system consists of multiple processes and these processes communicate by exchanging messages.
- Communication failures do not partition the network.
- A set of checkpoints of different processes form a consistent system state.
- The set contains exactly one checkpoint for each process.

After having identified the protocol building blocks and subsequently the high-level framework for protocol composition, we now consider a commonly used distributed dependable protocol to illustrate the applicability of our approach.

3.6 FDIR Protocol Composition

A fault tolerant system employing dynamic redundancy techniques achieves a desired dependability attributes in the presence of faults through a process of Fault Detection, fault Isolation and resource Reconfiguration, typically referred to as the (FDIR) paradigm. Fault diagnosis [7, 12] is a key component of this approach, requiring an accurate determination of the status of the system. Fault isolation component prevents a faulty unit from causing incorrect behavior in a non-faulty unit. We outline the basic aspects of the generic on-line fault diagnosis and FDIR algorithm [16] that we use to demonstrate our approach.

Fault Diagnosis and FDIR Algorithm

Fault diagnosis in the system is achieved through constant monitoring and exchange of data between different system nodes. Each node broadcasts its output value at frame end. Each node has a majority voter, which votes on the output data from all the nodes including its own output at the end of every frame. A non-faulty node can identify the sender of an incoming message, and can detect the absence or deviation from specified time window for an expected message. The determination of an error in a node ($\mathcal{X}$) is achieved by the other nodes in the system as they receive and analyze the incoming messages from $\mathcal{X}$ for errors.

We highlight basic steps which are essential to the overall correctness of the FDIR protocol.

Module Name: FDIR
Module Attributes

- All processors execute the same workload and determine the output value using a voting function.
- At the end of a frame, each node votes on the output data from all the nodes including its own output, and generates an error report.
- At the beginning of every frame, each node broadcast an error report message regarding each node in the system.
- Upon receipt of a round of error report, each node votes and collectively agrees upon the penalty count of each other nodes.
- Once this exact agreement is reached, the penalty count of an accused node is compared to the prespecified exclusion threshold.
- If the penalty count exceeds the threshold, each node then votes on the exclusion of an accused node from the operating set, and collectively (Byzantine agreement) agrees upon this decision.
- If an excluded nodes exhibit correct behavior, its penalty and reward counts are updated, and all other functional nodes agree upon these values.
- If the reward count of an excluded node falls below a predefined re-admission threshold, then its inclusion in the current operating set is collectively (Byzantine agreement) agreed upon by all other functional nodes.

We now relate these functions to protocol-building-blocks which are constituent to the hierarchical composition of the FDIR protocol.

A: Choosing the Building Blocks:

Through these identified functions/steps, it can be inferred that the following basic protocol-building-blocks are essentially needed. We discuss their role in the overall operation of fault detection, isolation and reconfiguration. The building blocks being used are:

- *Voting/Convergence Function* is used to (a) determine the output value of the processor, and detect any error in an incoming message, (b) compute the average value during interactive convergence (synchronization) phase, and (c) compute the majority value(s) during exact agreement phase.
- *Synchronization* is used to (a) synchronize various activities of all processors, and (b) implement atomic broadcast primitive.
- *Communication Primitives* (specifically, atomic broadcast primitive) is used to achieve the atomicity, order and bounded communication of message exchanges.
- *Agreement Function* (or interactive consistency algorithm (ICA)) is used to collectively agree upon (a) penalty count and/or reward count of an accused node, and (b) inclusion/exclusion of a node in/from the operating set of nodes.
- *Checkpointing Function* is used to establish recovery points to restart the process upon the initiation of reconfiguration of the operating set of nodes. This function may be implemented implicitly by using the frame boundaries from the synchronization function.

B: Outlining the Block Interactions:

Having identified the necessary blocks, Figure 5 then depicts the hierarchical composition of FDIR protocol utilizing the identified building blocks of redundancy management protocols.

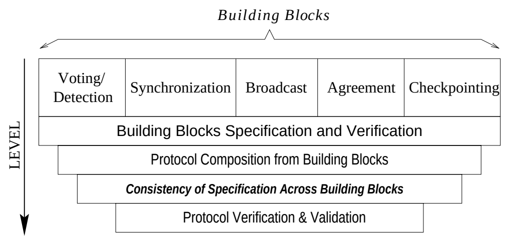

Building Blocks
Voting/ Detection
Synchronization
Broadcast
Agreement
Checkpointing
Building Blocks Specification and Verification
Protocol Composition from Building Blocks
Consistency of Specification Across Building Blocks
Protocol Verification & Validation
LEVEL

Figure 5: Composition of a FDIR Protocol

An important step in composing a protocol is to establish the consistency of specifications across various constituent blocks by demonstrating that the requirements of these building blocks are non-conflicting. Towards this objective, we identify the inter-dependencies of these building blocks in the overall FDIR protocol operation (See Figure 6).

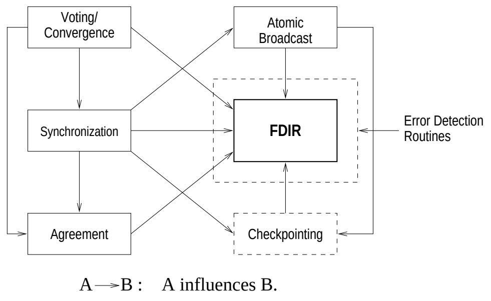


Figure 6: Inter-dependencies of Building Blocks

Since the influence paths in Figure 6 outline the interactions, our next step is to identify the exact condition or conditions a particular block imposes on another blocks. We summarize the conditions (for Byzantine fault coverage) for each block interactions in Table 1. Let us now consider an influence path from Figure 6, specifically Voting/Conv $\rightarrow$ Sync $\rightarrow$ At.Bcast $\rightarrow$ FDIR to elaborate on the nuances of block-interactions. As we see from Table 1 that the influence of atomic-broadcast block on the FDIR block is that the underlying system model must be synchronous. The synchronous nature of the system essentially requires the synchronization block. Voting/convergence block primarily influences the system configuration for the given fault coverage. Now, suppose that for the Chkptg $\rightarrow$ FDIR block-interaction, one chooses an asynchronous checkpointing algorithm to establish recovery points. A basic assumption in asynchronous checkpointing algorithms is that the message transmission delay is arbitrary but finite. However, from the correctness viewpoint, a protocol that is designed for an asynchronous system (weaker assumptions) will still be correct when executed in a synchronous system (stronger assumptions).

C: Issues in Block-Interactions:

We note that the effectiveness of the process of isolating and defining self-contained modules depends on capabilities to identify underlying direct and indirect dependencies between building blocks.

For illustration purposes, we focus on the compatibility of three specific building blocks namely, synchronization, atomic broadcast, and checkpointing, being used in FDIR composition. One of the requirements [7] of a consistent set of recovery points taken by the checkpointing operation is that there are no orphan messages[2]. As to be illustrated, the ability of a checkpointing operation to ensure that there are no orphan messages in a consistent set of recovery points is governed by properties and correctness of synchronization and atomic broadcast primitives. A checkpointing operation can be implemented using the frame boundaries from the synchronization function. In reality, it could be possible that at any global time, the clock reading of any two different processors in the system is within β of each other. Let at time instant T, each process is scheduled to take its checkpoints. Since clocks are not perfectly synchronized, the clocks of different processor will reach T within β of each other. Consider a particular case of a message exchange between two processors (See Figure 7).

Blocks Interactions	Attributes/Conditions
Voting/Conv $\rightarrow$ FDIR Voting/Conv $\rightarrow$ Sync Voting/Conv $\rightarrow$ Agrmt	$N = 3m + 1$ units N *identical* units voting cycle voting time $<$ arrival rate
Sync $\rightarrow$ At.Bcast Sync $\rightarrow$ FDIR Sync $\rightarrow$ Chkptg Sync $\rightarrow$ Agrmt	initially synchronized clocks bounded drift rate bounded clock-reading error $3m + 1$ processes
Agrmt $\rightarrow$ FDIR	$3m + 1$ participants $2m + 1$ disjoint paths $m + 1$ rounds of communication synchronous system model
At.Bcast $\rightarrow$ FDIR At.Bcast $\rightarrow$ Chkptg	synchronous system model bounded delays synchronized clocks
Chkptg $\rightarrow$ FDIR	un-partitioned network co-ordinated checkpoints

Table 1: Inter-Block Requirements

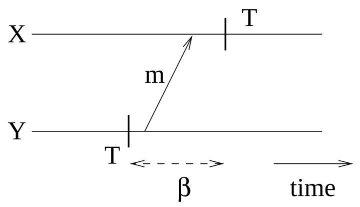


Figure 7: Checkpointing: Interaction of Processes

[2]There is no event for sending a message in a process Y succeeding its checkpoint, whose corresponding receive event in another process X occurs before the checkpoint of X that is contained in the set [7].

It is to note that processors are supposed to take checkpoints at time instant T as per their respective local clock. Here, processor Y sends a message m to X after establishing its checkpoints. Since there is no lower bound imposed on broadcast delay (it is only upper bounded by a constant Δ), in the event when $\Delta < \beta$, the message m can get delivered at X before X takes its checkpoint at time instant T. Under this situation, the set of checkpoints taken by X at time instant T will have an orphan message.

We emphasize that by modularizing and identifying specific requirements and dependencies of each individual building block, we can identify subtle cases (as the one discussed above) which might occur when building blocks are operating together.

D: Composition Guidelines Revisited:

After having discussed these inter-block interactions, we now refer back to category-based composition guidelines presented in Section 2.1 to highlight some important issues of this FDIR composition. Let us consider module interactions depicted in Figure 8 to discuss various issues of composition. For clarity purposes, we have considered module interactions between clock synchronization and atomic broadcast. The argument extends for other module interactions as well.

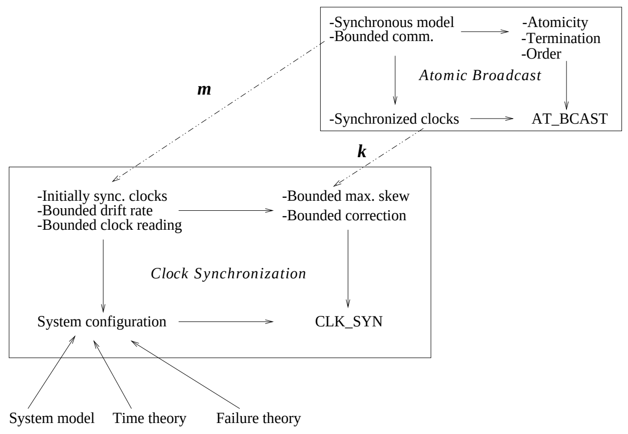


Figure 8: Composition of Synchronization and Atomic Broadcast Modules

We outline[3] formal theories of synchronization and atomic broadcast primitives. This also illustrates how export and import interfaces are defined. Theories of other system building blocks can be similarly described. Following this, we present FDIR composition utilizing theories of these system building blocks. We emphasize that once the basic formal specification and analysis of a particular building block is developed, associated axioms and formal theories of these blocks essentially remain unchanged over the protocol composition.

[3]For clarity purposes, we have opted not to use any particular syntax of a specification language.

```
clock_synchronization : theory
 IMPORTING time_model, system_model, failure_model
 % Specifications of semantic requirements
 begin assumption
 % declared as AXIOMS (See Section 3.2)
  -- initial skew : AX_1
  -- resynchronization period : AX_2
  -- bounded drift : AX_3
  -- initial synchronization : AX_4
  -- nonoverlap : AX_5
  -- clock reading errors : AX_6
 end assumption
 % Defines EXPORT interface
  -- bounded max skew : THEOREM_1
  -- bounded correction amount : THEOREM_2
end theory

atomic_broadcast : theory
 IMPORTING clock_synchronization,synchronous_system
 % Specifications of semantic requirements
 begin assumption
 % declared as AXIOMS (See Section 3.3)
   -- synchronized clocks : AX_1
   -- connected network : AX_2
   -- bounded transmission : AX_3
 end assumption
 % Defines EXPORT interface
   -- atomicity : THEOREM_1
   -- order : THEOREM_2
   -- termination : THEOREM_3
end theory
```

Module Atomic Broadcast (AB) utilizes services extended by Clock Synchronization (CS) to achieve atomicity and bounded communication of message exchange. Module AB imports *synchronization* function through export interface of CS module. Morphisms k and m are needed to establish the compatibility of these two modules. Consider morphism k which links import specification of AB module to export specification of CS block. As we mentioned, $k : SPEC1 \rightarrow SPEC2$ must be such that image of any axiom of SPEC1 is a theorem in SPEC2. In AB block, the fact that "the clocks of correct processors are approximately synchronized within a maximum allowable deviation" is specified as an axiom (Refer to theory outline presented below). For morphism k, the image of this axiom is essentially a theorem in CS module, i.e., we must prove in CS block that the maximum skew between any two good clocks are bounded (See Sections 3.2 and 3.3). Similarly, morphism m maps parameters of synchronous model of AB modules to CS module. Basically, this is an identity morphism. This readily follows from module attributes of Synchronous System and Synchronization presented in Sections 2.2 and 3.2, respectively.

To further elaborate, consider now composing Agreement block with the composite module out of AB and CS blocks. The specification R which describes the semantic requirements of Agreement module will import parameters from AB and CS block (Refer to Fig. 4). Also, we have to define morphisms to indicate compatibility between the parameters of the two modules. For example, we could define a morphism which maps *connectivity* requirements of Agreement module to *connected network* requirements of AB module. Similarly, another morphism maps synchronization attributes (Refer to Sections 3.3 and 3.4). As a note, a group membership protocol can also be modularly constructed utilizing system building blocks of clock synchronization and atomic broadcast. The formal theory of FDIR is described below.

```
FDIR : theory
 IMPORTING clock_synchronization, atomic_broadcast,
           agreement, membership, checkpointing
 begin assumption
 % Services of other system building blocks
 % declared as AXIOMS (See Sections 3.1-3.5)
  -- synchronized clock : AX_1
  -- atomicity of broadcast : AX_2
  -- termination of broadcast : AX_3
  -- bounded failure detection : AX_4
  -- agreement on processors' group-view : AX_5
  -- recognition of a processor in a group : AX_6
  -- a consistent system state for recovery : AX_7
 end assumption
 % Operational attributes
  -- voting cycle rate
  -- penalty count threshold
  -- reward count threshold
  -- error appearance rate
  -- error disappearance rate
 % Requirements of FDIR -- properties to be proven
  -- diagnosis of a faulty node : THEOREM_1
  -- agreement on node's exclusion : THEOREM_2
  -- agreement on node's inclusion : THEOREM_3
  -- reconfiguration operating nodes : THEOREM_4
end theory
```

We now define some of the morphisms needed for FDIR composition, and discuss how they can assist in establishing compatibility of modules, and in simplifying verification.

```
Morph_A : spec_FDIR -> spec_Agreement
 -- maps attributes
 -- maps assumptions
 -- maps functions
    Declare_Faulty(processor_set,round,processor_1,
    processor_2) -> ICA(processor_set,value_vector)
    Exclude(processor) -> ICA(processor_set,
                          accused_value_vector)
    Include(processor) -> ICA(processor_set,
                          approved_value_vector)
Morph_B : spec_FDIR -> spec_Voting
 -- maps attributes
    system_voting_cycle -> voter_processing_rate
    threshold_value -> error_appearance
    threshold_value -> disappearance_rate
 -- maps assumptions
 -- maps functions
    Detect_Error(output_values) ->
                Majority_Vote(value_vector)

Morph_C: spec_FDIR -> spec_Synchronization
 -- maps attributes
    system_voting_cycle -> resynchronization_rate

Morph_D : spec_Agreement -> spec_Voting
 -- maps attributes
 -- maps assumptions
 -- maps functions
    ICA(processor_set, value_vector) ->
                 Majority_Vote(value_vector)
```

E: Conformance Aspects of Morphisms over Composition:

Let us consider a hypothetical situation where we change few attributes/actions in some modules for them to become operationally conflicting. In FDIR protocol, system voting cycle is the frame rate at which nodes exchange data for voting. Suppose, we decide to decrease the rate at which systems participate in clock synchronization. Since we want that each processor to be executing the same workload and broadcasting its output at frame end, we have to accordingly decrease the system voting cycle to make it work in lock-step manner. In the case of reducing the system voting cycle, if the error is transient in nature and system penalty count threshold is high, the node gets isolated after a long time and hence error isolation latencies are very large. This would thus reduce the system reliability. This conflicting design choice can get flagged by Morph_C mapping FDIR to Synchronization while translating properties incorporating system voting cycle and system resynchronization period to FDIR's operating conditions for handling transient errors.

Moreover, in order for a processor to declare another processor faulty, it executes an interactive consistency algorithm (ICA) using the error report collected over the beginning of the frame. The correctness property of the diagnosis procedure (`Theorem_1` of FDIR theory) can be established by translating properties of interactive consistency/Byzantine agreement along Morph_A which maps specification FDIR to Agreement.

4 Discussion and Conclusions

The key idea presented in this paper is that if a library of – system, fault, communication – models and building blocks based on them can be formulated, then these elements subsequently aid in systematic and hierarchical development of the formal models of dependable distributed protocols. We have presented our initial approach towards exploiting category theory for modular protocol composition, and have discussed how by defining export and import interfaces of basic modules, and morphisms linking two different modules, a larger or more complex protocol can be formally composed and verified.

We envision that a modular approach such as the one we have proposed in this paper would facilitate easier design process and formal treatment of protocols with stricter real-time and dependability requirements. Such building blocks will be useful to system designers because they will permit thoroughly (and rigorously) tested formal theories of required system and component behavior, and will support system design decisions and modification. One of the interesting viewpoint is to investigate techniques to ensure the consistency of varied requirements of building blocks across different levels. Further research in this context would require detailing and defining external specifications of building blocks to facilitate such mechanisms.

References

[1] R. Alur, et al., "MOCHA: Modularity in Model Checking." *LNCS 1427*, pp. 521–525, Springer-Verlag, 1998.

[2] A. Arora, S. Kulkarni, "Component Based Design of Multitolerance," *IEEE Trans. on Software Engineering*, vol. 24, no.1, pp. 63–78, 1998.

[3] E.M. Clarke, J.M. Wing, et al., "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys*, vol. 28, No. 4, pp. 626–643, Dec. 1996.

[4] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 2 – Module Specifications and Constraints*, vol. 21 of *EATCS Monograph on Theoretical Computer Science*, Springer-verlag, 1990.

[5] B. Garbinato, R. Guerraoui, "Flexible Protocol Composition in BAST," *ICDCS–18*, pp. 22–29, 1998.

[6] M.A. Hiltunen, R.D. Schlichting, "An Approach to Constructing Modular Fault-Tolerant Protocols." *SRDS–12*, pp. 105–114, Oct. 1993.

[7] P. Jalote, *Fault Tolerance in Distributed Systems.* Prentice Hall, 1994.

[8] X. Liu, et al., "Building Reliable, High-Performance Communication Systems from Components." *Operating Systems Review*, 34(5), pp. 80–92, Dec. 1999.

[9] P. Michel, V. Wiels, "A Framework for Modular Formal Specification and Verification." *Proc. of FME'97*, 1997.

[10] S. Misra, et al., "Consul: A Communication Substrate for Fault-Tolerant Distributed Programs." *Distributed Systems Engineering*, 1(2), pp. 87–103, 1993.

[11] D.E. Rydeheard, R.M. Burstall, *Computational Category Theory*, Prentice Hall, 1988.

[12] M. Singhal, N.G. Shivratri, *Advance Concepts in Operating Systems.* McGraw-Hill, 1994.

[13] P. Sinha, N. Suri, "Identification of Test Cases Using a Formal Approach." *FTCS-29*, pp. 314–321, 1999.

[14] N. Suri, P. Sinha, "On the Use of Formal Techniques for Validation." *FTCS-28*, pp. 390–399, 1998.

[15] R. van Renesse, K. Birman, S. Maffeis, "Horus: A Flexible Group Communication System." *Communication of the ACM*, 39(4), pp. 76–83, April 1996.

[16] C. Walter, P. Lincoln, N. Suri, "Formally Verified On-Line Diagnosis." *IEEE Trans. on Software Engineering*, SE 23(11), pp. 684–721, Nov. 1997.