On Simplifying Modular Specification and Verification of Distributed Protocols*

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

Computer systems supporting high assurance and high consequences applications typically utilize dependable distributed protocols to manage system resources and to provide sustained delivery of services in the presence of failures. The inherent complexity entailed in the design and analysis of such protocols, is increasingly necessitating the use of formal techniques in establishing the correctness of the protocol level operations. Exploiting modular design aspects appearing in most dependable distributed protocols, we have introduced techniques utilizing concepts of category theory for constructing formal library routines of a set of constituent functional primitives, and their use in establishing the correctness of the protocol operation. In this paper, we develop on our proposed category-theory-based approach for modular composition through formulating (a) a group membership protocol which can also form the next hierarchical building blocks for other dependable protocol operations, and (b) a checkpointing protocol utilizing the group membership function as one of its building block. Subtleties in building-block interactions and their influence on the overall correctness of the composite protocols are also highlighted.

1 Introduction

High assurance systems such as flight control systems, unmanned air vehicles, military command and control systems, etc. typically rely on group communication protocols to co-ordinate the system activities in the presence of failures. Due to complex protocol operations and corresponding large operational state space, simulation and prototyping techniques are facing growing limitations in establishing the correctness of protocol level operations. Exploiting formal methods [3] support for state exploration over the formal verification based analysis of operations of a given protocol, in [23, 21] we introduced a formal-methods-based approach to specifically identify pertinent test cases to guide and supplement the existing validation process. Over these studies, we observed that most dependable protocols are conceptually constructed using similar functional primitives or variations of these primitives, and perhaps their formal specification could be reused across different protocols. To support this viewpoint, in [20] we presented our initial approach towards exploiting category theory for modular composition of dependable distributed protocols. We have shown how by defining external interfaces of basic modules, and morphisms linking two different modules, a larger or more complex protocol can be formally composed and consequently verified in a compositional sense.

In this paper, we extend our category-theory-based approach introduced in [20]. Specifically, we (a) highlight the capabilities of our approach through a case study of group communication protocols, and (b) show how a complex protocol could be easily formalized if guidelines were provided as per the reuse of the formal specification and verification of individual functional primitives of the chosen protocol.

We first discuss related work in the area of modularization in order to distinguish our approach towards protocol modularization.

Related Work: Modularization is a well-known technique for simplifying complex software systems [1, 2, 6, 8, 9, 10, 11, 13, 14, 15, 17, 24]. We have observed that most of the approaches [8, 10, 17, 24] focus on *implementation* aspects of the composition of a protocol from micro-protocols developed for specific services. Among other approaches [1, 2, 6, 9, 11, 14, 15] have provisions for formal reasoning to ascertain configurations against system specifications. Our proposed approach [20] utilizes category-theory-based concepts to define interfaces for building block protocols and operations to complex protocol specifications. We envision that the correctness of the composite protocol level operations can be established utilizing the proof constructs being developed for basic protocol modules.

The organization of the paper is as follows. Section 2 provides a background discussion of our proposed

*Supported in part by NSERC, FRDP–Concordia University (Sinha) & TFR, Saab, NSF CAREER CCR 9896321 (Suri)

category-theory-based formal framework for modular composition. Sections 3.1 through 3.3 outlines the basic building blocks of group communication protocols. Section 4 presents a hierarchical composition of a checkpointing protocol to illustrate our modular approach. We conclude with a discussion in Section 5.

2 Modular Composition & Verification

In [20] we introduced a formal framework which utilizes concepts of category theory to facilitate a rigorous and consistent composition out of system building-block protocols. Here, we first provide a brief overview of the proposed category theory based modular composition framework prior to detailing the overall approach based on it.

2.1 Category-Based Composition

We have adapted *calculus of modules* based on categorical concepts [7, 16, 18] for simplifying formal verifications. We first define a few general terms; for details, the reader is referred to [7, 18].

- **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 $m : SPEC1 \rightarrow SPEC2$ is a map from the sorts and operations of one specification to the sorts and operations of another such that (a) axioms are translated to theorems, and (b) source operations are translated compatibly to target operations.

Module specifications are defined by utilizing the notion of push-out operation from 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 1(a)).

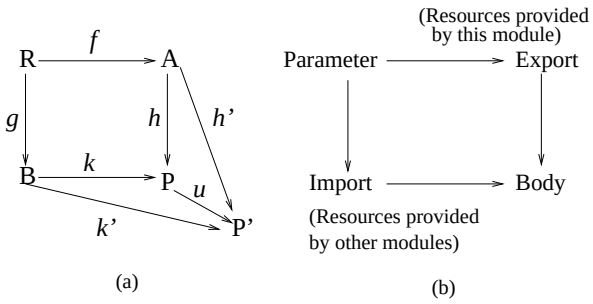

Figure 1: (a) Push-out, and (b) Module Interfaces

Formally, given specification morphism $f : R \rightarrow A$ and $g : R \rightarrow B$, a specification P together with specification morphisms $h : A \rightarrow P$ and $k : B \rightarrow P$ 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 $u : P \rightarrow P'$ 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 (See Figure 1(b)). In other words, specifications A and B correspond to guarantees and assumptions, respectively. 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: To capture module interactions, our proposed composition scheme allows two modules to be interconnected via export and import interfaces. The push-out of two modules is the resulting specification of the composed module. Figure 2 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.

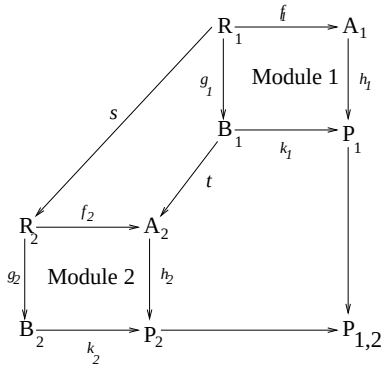

Figure 2: Composition of Two Modules

For 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. We refer the reader to [20] for further details.

Our proposed category-theoretic-framework [20] provides assistance to a protocol designer to choose constituent modules and test for their compatibility as part of the selection process. Figure 3 depicts our general approach for modular composition and verification of dependable distributed protocols. We begin with highlighting the salient features of basic building blocks mentioned on the top most level in Figure 3.

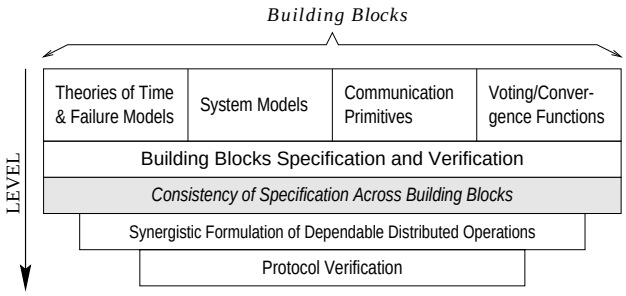

Figure 3: A General Framework

2.2 Basic Building Blocks

System Models

In a distributed system model, the notion of synchrony relates to assumptions made about the time bounds on the performance of the system. We characterize pertinent aspects of synchrony in the module given below. We will utilize this format of "module" description throughout the paper.

Module Name: Synchronous System[1]
Module Attributes:

- 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 has a local clock with known bounded rate of drift $\rho \geq 0$ with respect to real-time.
- 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.

Failure Models

A *failure model* specifies how a faulty component can deviate from its service specification. 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.

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

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. Thus, a specification of a datagram service reflecting bounded communication under different failure assumptions can be specified.

Voting/Convergence Functions

An important issue in distributed computing is to ensure that interacting nodes adopt identical or close values for system parameters. Two main approaches, namely consensus and convergence differ in the decision procedure and the chosen voting function(s).

2.3 Basic Blocks → 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 group communication 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 with 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.

Having discussed our category theory based modular approach for protocol composition, we now elaborate on our approach by addressing classes of protocols used for *group communication* services.

3 Group Communication Services

Application servers replicate the system state and use group communication protocols to co-ordinate their activities in the presence of processor or communication failures. The empirical formulation of a generic group communication protocol involves the following procedures: **(a)** establishing and maintaining a common time base among distributed functional units and between computational tasks (*clock synchronization*),

(b) providing consistent information to multiple processors in the distributed system ensuring message ordering and guaranteed delivery (*atomic broadcast*), and **(c)** ensuring up-to-date state information at each group member (*group membership*).

We categorize basic functions that are typically used across a variety of group communication protocols. For each block, we (informally) outline primary attributes[2]; for details, we refer the reader to [4, 12, 19].

3.1 Building Block 1: Synchronization

In a distributed environment, to establish and maintain a consistent system-wide time base, each processor periodically executes a protocol that involves exchanging clock values with the other processors, computing a reference value, and adjusting its local clock to reflect the consensus.

Module Name: Clock Synchronization
Module Attributes:

- The non-faulty clocks are initially synchronized to some constant quantity.
- The non-faulty clock should not drift by a rate greater than the specified drift-rate.
- Re-synchronization signals must occur within a specified period.
- Synchronization periods should not overlap.
- The error in reading clock values of non-faulty processors is bounded by a constant.

Approach:

- Periodically, the processors decide that it is time to re-synchronize their clocks. Each processor reads the clocks of the other processors, forms a "fault-tolerant average" of their values, and sets its own clock to that value.

Requirement:

- The maximum skew between any two good clocks must be bounded.
- A non-faulty processor's clock is adjusted by a small amount during each re-synchronization.

3.2 Building Block 2: Atomic Broadcast

An important objective in distributed system is to provide consistent information to multiple processors in the system. Since the basic communication primitive supported by a network is one-to-one communication, broadcast services must make use of this basic primitive.

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, Δ.

Approach:

- To execute A-broadcast m, a process simply Reliable-broadcast m, i.e., it sends m to all its neighbors in the network.
- When a process R-delivers m, it schedules its A-delivery at local time $T + \Delta$.

Requirements:

- *Atomicity:* if any correct processor delivers a message at time T on its clock, then that message was initiated by some processor and is delivered by each correct processor at time T on its clock.
- *Order:* all correct processors deliver their messages in the same order.
- *Termination:* every message whose broadcast was initiated by a correct processor at time T on its clock is delivered by all correct processors at time $T + \Delta$ on their own clocks.

At this stage we have presented two building blocks namely, *clock synchronization* and *atomic broadcast*. In the next section we will show how the *group membership*, although a building block by itself, can be hierarchically composed of other identified building blocks.

3.3 Building Block 3: Group Membership

We now introduce the final building block component for distributed dependable protocols, i.e., the *group membership* primitive. Group membership protocols are used to ensure that the state information stored at each group member remains up-to-date and that at any time, all group members see the same state information – despite information propagation delays and failures.

Module Name: Synchronous Group Membership
Module Attributes:

- The network remains connected even upon failures.
- Atomic broadcast service must be implemented to communicate between processors.

Approach:

- A processor j that starts at time T, broadcasts a "new-group" message time-stamped T to all other processors to form a new group.
- On receiving "new-group" message, each processor broadcasts a "present" message with its identifier and its willingness to join a new group. The set of processors that broadcast "present" messages for time $S = T + \Delta$ is termed *processor membership as of view time S*.
- To ensure consistency of membership view, each member of the group broadcast "present" message at a defined "membership check" time.

Requirements:

- *Agreement on group membership:* If p and q are joined to the same group and are both alive then their membership views are identical.
- *Recognition:* If p is alive and joined to a group then its *id* will be included in its membership view.
- *Bounded failure detection delays:* There exists a time constant d_f such that if a processor belonging to group g fails at time t then, by time $t + d_f$, all members of g that stays correct in the interval $[t, t + d_f]$ will join a group G' that does not contain p.
- *Bounded join delays:* There exists a time constant d_j such that if a processor starts at time t then, by time $t + d_j$, it will join a new group along with every other processor q that stays correct in the interval $[t, t + d_j]$.

[2] In module descriptions, the attributes correspond to the assumptions (Specification B), and the requirements correspond to the guarantees (Specification A) (See Figure 1(a)).

The *synchronous membership protocols* of [5], which depend on (a) synchronous atomic broadcast service, (b) clocks internally synchronized to a maximum deviation of δ, and (c) an unreliable datagram service, accurately identify the correct processes, and satisfy the above stated timeliness properties. It is interesting to see how the group membership can be modularly composed utilizing system primitives and building blocks 1 and 2 of the prior sections. A composition involving these different categorized modules and their associated morphisms is depicted in Figure 4.

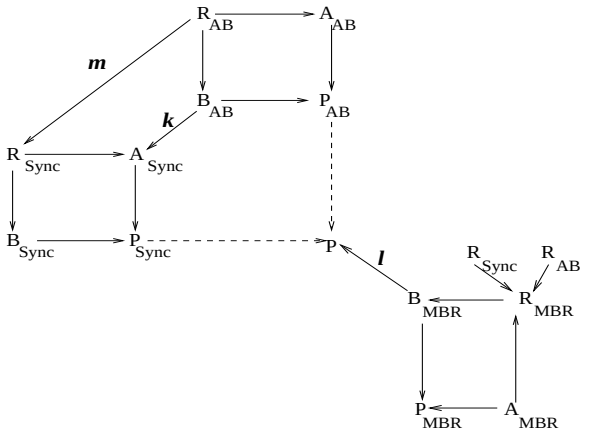

Figure 4: Composition of a Synchronous Membership Protocol

Composition Guidelines: Let us consider module interactions between clock synchronization and atomic broadcast depicted in Figure 4. We outline[3] formal theories of synchronization and atomic broadcast primitives. This also illustrates how export and import interfaces are defined (See Figure 5). In these module definitions, "function" specifies a particular approach needed to realize the implementation of that specific operation.

Module Atomic Broadcast (AB) utilizes services extended by Clock Synchronization (Sync) to achieve atomicity and bounded communication of message exchange. Module AB imports *synchronization* function through export interface of Sync 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 Sync 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 (an assumption in Import part) (Refer to theory outline in Figure 5). For morphism k, the image of this axiom is essentially a theorem in Sync module (a requirement in Export part), i.e., we must prove in Sync block that the maximum skew between any two good clocks are bounded. Similarly, morphism m maps parameters of synchronous model of AB modules to Sync module. This readily follows from module attributes of Synchronous System and Synchronization presented in Sections 2.2 and 3.1, respectively. Specification P is the co-limit of complete specifications P_{Sync} and P_{AB}.

[3]The development of complete theories of these building blocks in Specware [22] and their usage in compositional verification is an ongoing work.

Parameter (R)
System configuration
Time and failure models
Assumptions
un-partitioned network
bounded transmission delay
initial clock skew
bounded drift
initial synchronization
resynchronization period

Import (B)
Datagram function
Clock reading mechanism
bounded communication
bounded clock reading error
Fault-tolerant averaging function
precision enhancement (closeness)

Export (A)
Clock Synchronization function
bounded maximum skew
bounded clock correction amount

Body : Clock Synchronization

Parameter (R)
System configuration
Time and failure models
Assumptions
un-partitioned network
bounded transmission delay

Import (B)
Datagram function
bounded communication
Clock Synchronization function
bounded maximum skew
bounded clock correction

Export (A)
Atomic Broadcast function
atomicity
termination
order

Body : Atomic Broadcast

Figure 5: Categorized Theories of Clock Synchronization and Atomic Broadcast

Compositional Verification Aspects – Let us consider composing a membership block with the composite module of AB and Sync block. The specification R_{MBR} (See Figure 6) which describes the semantic requirements of Membership module is supported by parameters from AB and Sync blocks. The correctness properties of the protocol's action of broadcasting a message at regular intervals, joining a group and maintaining the view of membership can be decomposed into correctness properties of datagram services, synchronized clocks, and atomic broadcast, and then they get translated along morphisms (e.g., l) to prove that the membership requirements are met.

To illustrate this fact, we consider an example of "agreement on history" property of membership protocol. Suppose, p and q are correct processors during a given time interval. If during that interval, p and q are joined to a common group g and the next group

Parameter (R)

Synchronous communication network
(supported by "parameter" part of Sync. and Broadcast)

Import (B)

Atomic Broadcast function
(Imported via composite specification)
atomicity and terminiation property

Export (A)

Membership function
agreement on group membership
join detection
failure detection

Body : Group Membership

Figure 6: Specification of Group Membership Protocol

joined by these two processors after leaving g are g_1 and g_2, respectively, then groups g_1 and g_2 are the same. Since *time* is used to uniquely identify successive processor groups, the *termination* and *atomicity* properties of broadcast service used by a joining processor compels all *correct* processors to agree on a new view time as well as on the successive membership check instances.

4 Modular Composition of a Co-ordinated Checkpointing and Recovery Protocol

After having identified the protocol building blocks (Sections 3.1 and 3.3) and subsequently the high-level framework for protocol composition, we now consider an actual distributed protocol to illustrate the applicability of our approach. A checkpointing protocol is chosen as this represents a large class of distributed protocols widely used in both theory and practice.

Checkpointing [12, 19] 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. We consider an approach for checkpointing and rollback recovery which utilizes synchronized clocks as outlined in [12]. We highlight basic steps which are essential to the overall correctness of the chosen protocol.

Module Name: Checkpointing
Module Attributes

- All processes communicate by exchanging messages through communication channels.
- Communication failures do not partition the network.
- The clocks of the different processors are approximately synchronized within a maximum allowable deviation β.
- The message delay is bounded by a constant δ.
    - With synchronized clocks and bounded message delay, if a message is sent by a process in the time period (local clock) $[T - \beta - \delta, T]$, then it will be delivered to the destination by time $T + \delta + \beta$ (local clock).

Approach:

- All co-operating processes checkpoint periodically, each with the same period π. It is assumed that $\pi > \beta + \delta$.
- Prior to establishing its k^{th} checkpoint, a process does not consume any message that is sent by a process after establishing its k^{th} checkpoint.
- A process checkpoints its own local states, and logs the messages it sends or receives.
    - The senders log the messages they send during the interval $[T - \beta - \delta, T]$.
    - A receiver logs any message it receives in the interval $[T, T + \beta + \delta]$.

Requirement:

- A set of checkpoints of different processes form a consistent system state, i.e., orphan and lost messages are not present.

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

A: Choosing Building Blocks

Checkpointing protocols utilizing synchronized clocks would essentially require the building blocks of group communication services, namely synchronization, broadcast and membership. **Synchronization** is used to (a) synchronize various activities of all processors, and (b) implement atomic broadcast primitive, **communication primitives** (specifically, atomic broadcast) is used to achieve the atomicity, order and bounded communication of message exchanges. and finally, **membership** is essentially needed to identify members in a group to form a consistent global state. These building blocks inherently assume the presence of synchronous communication network primitives for bounded communication between two processes.

B: Outlining the Block Interactions

Having identified the building blocks and their role in achieving the overall objectives of the checkpointing operation, we present the hierarchical composition of the checkpointing protocol using the building blocks of group communication services, as shown in Figure 7.

In order to establish the consistency of specifications across the various constituent building blocks, it is important to identify inter-dependencies of these basic building blocks (as depicted in Figure 8), and highlight the exact condition or conditions a particular block imposes on another blocks. As the modularization approach help us identify the specific attributes of each

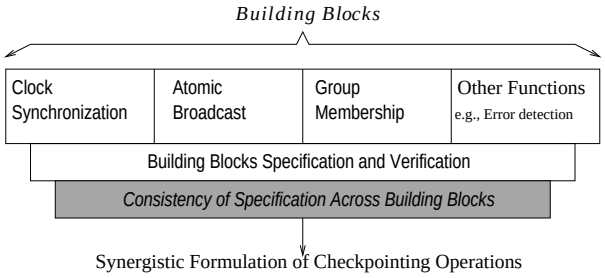

Figure 7: Hierarchical Composition of a Co-ordinated Checkpointing Protocol

block, we summarize the conditions for each block interactions in Table 1.

Blocks Interactions	Primary Attributes
Sync. → Broadcast Sync. → Membership Sync. → Checkpoint	synchronized clocks bounded drift rate & clock reading error
Broadcast → Membership Broadcast → Checkpoint	synchronous system bounded delays synchronized clocks
Membership → Checkpoint	unpartitioned network synchronized processors bounded communication atomicity and ordering

Table 1: Inter-Block Requirements for Checkpointing

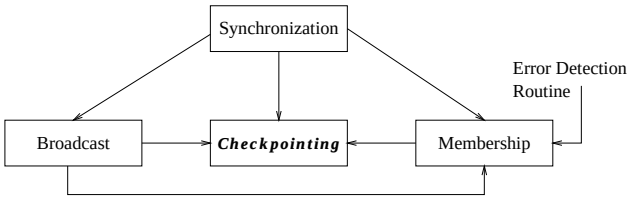

Figure 8: Inter-dependencies of Building Blocks of Checkpointing Protocol

The interactions between building blocks are depicted in Figure 8. An influence path, such as Synchronization → Broadcast → Membership → Checkpointing, depicts a form of interactions among these four building blocks. We illustrate how modular composition assist in addressing these dependencies and identifying any governing conditions. By the properties of the synchronous membership protocol, each processor has an up-to-date state information of all group members. Suppose, a failure is detected in a process by a processor P. As a result, P must timely broadcast this event to all other processes to stop all execution and initiate a roll back to a validated checkpoint. These facts can be asserted by the atomicity and termination properties of atomic broadcast. The influence of atomic broadcast block on the membership block is that the underlying system model must be synchronized. This constraint gets satisfied by the synchronization block.

C: Issues in Block Interactions

We further elaborate on inter-dependencies by considering the properties of broadcast primitive and the nature of clock synchronization that can influence the working of checkpointing operation. Clocks of all processors are tightly synchronized with respect to each other with a maximum skew of β. With approximately synchronized clocks and no lower bound imposed on broadcast delay, it is possible that a set of checkpoints taken by a processor may contain an orphan message. To avoid inclusion of orphan messages in a checkpoint, the checkpointing operation ensures that a process does not consume any message delivered during β before establishing its checkpoint. To achieve this, all messages sent by a process between its k^{th} checkpoint and $k+1^{th}$ checkpoint are tagged with k to indicate the interval in which they are sent. Only messages with tag $k-1$ are included in the k^{th} checkpoint. The correctness of this technique is guaranteed by the existence of a common time base by synchronizing the clocks of all processors in the system. These subtle dependencies of building blocks can be highlighted by modularizing and defining morphisms portraying these specific requirements.

D: Nuances of Morphisms over Modular Composition

We have illustrated various issues involved in category-based composition of protocols while discussing group membership building block. In this section, we present an informal composite specification of checkpointing protocol (See Figure 9), and then define some morphisms needed for a consistent composition out of identified system building blocks.

We now define some of the morphisms and discuss how they can assist in simplifying formal verification. For discussion purposes, we consider three different morphisms, namely $Morph_A$, $Morph_B$ and $Morph_C$ linking checkpoint specification to membership, atomic broadcast and synchronization specifications, respectively. That is, $Morph_A : Checkpoint \rightarrow Membership$, $Morph_B : Checkpoint \rightarrow Broadcast$, and $Morph_C : Checkpoint \rightarrow Synchronization$.

First, via $Morph_A$ membership function establishes the consistent view of membership of each processes,

Parameter (R)

Synchronous network
(supported by parameter part of Sync. and Broadcast)
Operational Attributes
checkpointing interval

Import (B)

Clock Synchronization function
synchronized clocks

Atomic Broadcast function
atomicity and termination property

Membership function
agreement on group membership
join detection
failure detection

Export (A)

Checkpointing function
a consistent global set of checkpoints

Body : Checkpointing Operation

Figure 9: Specification of Checkpointing Protocol

i.e., determines participating processes. As mentioned, in order to establish a consistent checkpoint, it is important that sender and receiver log any messages sent or received during the specified interval to avoid orphan or lost messages. The correctness of this property can be established by translating correctness properties of atomic broadcast operation along $Morph_B$. $Morph_C$ is needed to ensure that the frame boundary of each processor is synchronized, and all co-operating processor establish their checkpoints within a specified clock skew. The constraints on the checkpointing interval (π) is governed by the maximum allowable transmission delay (δ) and the maximum skew (β) between clocks of two correct processes, i.e, $\pi \geq \delta + \beta$. This also allows the detection of communication failure between checkpoints. These properties are translated along morphisms $Morph_B$ and $Morph_C$.

E: Perspectives on Tool-Supports for Composing Modules

We are currently specifying building blocks of group communication protocols in Specware [22]. Specware is a system for the specification and formal development of software. Among other operations available for combining specifications in Specware, we are exploiting the use of *import* and *colimit* operations. In order to use the colimit operation to combine specifications, we first indicate how they are to be related by exhibiting morphisms between them (called diagram of specifications). A *diagram* is a directed multigraph (allows more than one arc between nodes) whose nodes are labeled with specifications and whose arcs are labeled with morphisms. The colimit operation is then applied to a diagram of specifications linked by morphisms. The colimit contains all the elements of the specifications in the diagram, but elements that are linked by arcs in the diagram are identified in the colimit. Another basic operation of our interests in Specware is *refinement/interpretation*. This feature allows transformation of high-level task-descriptive specifications to low-level implementation-oriented specifications, and eventually to code generation.

As discussed, category-based formalization of basic building block-protocols permit re-usability of these basic formal modules. Moreover, for any configuration of building blocks over a protocol composition, morphisms also provide a direct capability of tracing desired feasible path of any variable, attribute, or action.

F: Caveats/Limitations

We note that selecting an appropriate set of building blocks for a given protocol can sometimes be difficult due to subtleties associated with fault-tolerance and timing attributes. We acknowledge some of the caveats/limitations of our proposed approach.

- It is possible that two initially non-conflicting modules may end up being semantically or operationally conflicting at some arbitrary implementation details which are not apparent at the high level of specification. Determining, *a priori*, how much (and also how often) implementation details will be needed to succinctly capture all subtleties of module interactions is a difficult problem and a topic of our on-going research.
- The choice of which building blocks to include, when composing a given protocol, and the parameterization of their respective formal theories are very much an intuitive process. In order to ensure a viable composition, the user needs to selectively choose building blocks which would satisfy varied functional, temporal and dependability requirements. Inability to identify subtle indirect dependencies among building blocks is the major cause for these problems.
- As expected, if a selected module cannot be added to an existing configuration to achieve the desired property, then that block must be modified accordingly. Moreover, for particular configurations of protocol-building-blocks, formal analyses may be required to determine the weakest properties that still can guarantee overall correctness of the protocol.

5 Discussion and Conclusions

For protocol level operations, the proposed category-theory based formal framework serves as a design tool for composing dependable distributed protocols. With our focus on group communication protocols, we have identified basic building blocks which are inherent to providing group communication services. As each module specification explicitly defines export and import interfaces, this scheme, depending on identified inter-dependencies between blocks, suffices in dealing with protocol interactions. It is important to note that morphisms linking different modules are effective means of highlighting any conflicts arising over composition. They essentially pinpoint properties which must be observed across different modules. As we can decompose global properties into small lemmas provable in different modules and translate these "local" properties along morphism to establish desired properties, we not only have a capability of performing basic verification, but also a design tool to perform speculative changes at the protocol and implementation level and observe the impact.

As systems in future grow more complex with stricter dependability specifications, the design of the protocol and, most importantly, its formal reasoning would necessitate a modular or a hierarchical approach to protocol composition. We envision that a modular approach such as the one we have proposed would facilitate modular composition of future protocols and their formal treatment. Such building blocks will be useful to system designers for their capability of specifying and facilitating rigorously tested (as well as pre-tested) formal theory modules of required system and component behavior, and also supporting system design decisions and modifications. Our present effort in expanding the proposed framework involve detailing and defining external specifications of building blocks to facilitate semi-automated process to establish consistency across building blocks.

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 Soft. Engg.*, 24(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] F. Cristian, "Understanding Fault-Tolerant Distributed Systems." *Comm. of the ACM*, 34(2), pp. 57–78, Feb. 1991.

[5] F. Cristian, "Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems." *Distributed Computing*, vol. 6, pp. 175–187, April 1991.

[6] R. De Prisco, et al., "Building Blocks for High Performance and Fault-Tolerant Distributed Systems." Details Available at http://www.lcs.mit.edu/research/projects, 1999.

[7] 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.

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

[9] T. Henzinger, et al., "You Assume, We Guarantee: Methodology and Case Studies." *Proc. of CAV'98*, July 1998.

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

[11] J. Hooman, *Specification and Compositional Verification of Real-Time Systems.* LNCS 558, Springer Verlag 1991.

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

[13] E. Juan, J.J.P. Tsai, *Compositional Verification of High-Assurance Systems*, Kluwer Academic Publisher, 2000.

[14] I. Keidar, R. Khazan, N. Lynch, A. Shvartsman, "An Inheritence-Based Technique for Building Simulation Proofs Incrementally." *Proc. of ICSE–22*, pp. 478–487, 2000.

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

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

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

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

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

[20] P. Sinha, N. Suri, "Modular Composition of Redundancy Management Protocols in Distributed Systems: An Outlook on Simplifying Protocol Level Formal Specification and Verification." *Proc. of ICDCS-21*, pp. 255–263, 2001.

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

[22] Y. V. Srinivas and Richard Jullig, "Specware(TM): Formal Support for Composing Software," *Proc. of the Conference on Mathematics of Program Construction*, Kloster Irsee, Germany, July 1995. Also as Kestrel Institute Technical Report KES.U.94.5.

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

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