

ANL/DIS/CP-94400  
CONF-9709167-  
M97 054216

A Layered Architecture for Critical Database Design<sup>1</sup>  
AFCEA - DOD Database Colloquium '97  
September 8-10, 1997  
San Diego, CA

RECEIVED  
SEP 22 1997  
OSTI

G. H. Chisholm and C. E. Swietlik  
Decision and Information Sciences Division  
Argonne National Laboratory

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

The submitted manuscript has been authored by a contractor of the U. S. Government under contract No. W-31-109-ENG-38. Accordingly, the U. S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

1. Work supported by the U.S. Department of Defense under interagency agreement, through U.S. Department of Energy contract W-31-109-Eng-38.

## DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

## **DISCLAIMER**

**Portions of this document may be illegible  
electronic image products. Images are  
produced from the best available original  
document.**

A Layered Architecture for Critical Database Design<sup>1</sup>  
AFCEA - DOD Database Colloquium '97  
September 8-10, 1997  
San Diego, CA

G. H. Chisholm and C. E. Swietlik  
Decision and Information Sciences Division  
Argonne National Laboratory

### ABSTRACT

Integrity, security, and safety are desired properties of database systems destined for use in critical applications. These properties are desirable because they determine a system's credibility. However, demonstrating that a system does, in fact, preserve these properties when implemented is a difficult task. The difficulty depends on the complexity of the associated design. We explore architectural paradigms that have been demonstrated to reduce system complexity and, thus, reduce the cost associated with certifying that the above properties are present in the final implementation.

The approach is based on the tenet that the design is divided into multiple layers. The critical functions and data make up the bottom layer, where the requirements for integrity, security, and safety are most rigid. Certification is dependent on the use of formal methods to specify and analyze the system. Appropriate formal methods are required to support certification that multiple properties are present in the final implementation. These methods must assure a rigid mapping from the top-level specification down through the implementation details. Application of a layered architecture reduces the scope of the design that must be formally specified and analyzed.

This paper describes a generic, layered architecture and a formal model for specification and analysis of complex systems that require rigid integrity, security, and safety properties.

### 1. INTRODUCTION AND BACKGROUND

The following database example is used throughout this paper to illustrate the points being discussed. The database supports a Battle Management/Command, Communications, and Control (BM/C<sup>3</sup>) element of a ballistic missile defense system. To support requirements for graceful degradation and devolution, the system is duplicated and dispersed geographically (for this example, at three separate locations). Each BM/C<sup>3</sup> subelement has collocated sensors. Thus communication is required to ensure that data are replicated among the three elements. This system is designed to destroy valid threats and only valid threats. Because of the nature of this system, it requires a multilevel, secure database (i.e., top secret, secret, confidential, and unclassified), with ultra-high integrity and safety constraints. The database component requires certification with respect to critical functionality, security, and safety. Such certification is quite expensive and is exacerbated by the complexity of the system.

---

1. Work supported by the U.S. Department of Defense under interagency agreement, through U.S. Department of Energy contract W-31-109-ENG-38.

This paper describes a paradigm for complexity reduction that will reduce the effort associated with system analysis and certification (i.e., quality assurance). The approach is based on developing a simple, understandable, and composable set of models and a mapping between them and an architecture.

## QUALITY ASSURANCE AND CERTIFICATION

The following conceptual statement describes the requirements of such a critical database:

... provide typical database functions (i.e., create, read/modify, and delete) without compromise of secure information and in support of critical functions. No data must be corrupted nor lost.

This statement consists of two types of requirements: *functional* and *design*. The functional requirements (e.g., to create) can be divided into subfunctions. The design requirements (i.e., to not compromise secure information, implying a degree of protection as high as reasonably attainable) permeate the entire system design; that is, they include both critical properties (e.g., security or safety) and operational properties (e.g., being able to operate under hostile environmental conditions). In the approach to the hierarchical design described here, each layer is analyzed for completeness with respect to functionality. Design requirements (e.g., a multilevel security policy) are either ascribed to a capability external to the database or to the underlying system resources.

Designers of critical systems strive for a high degree of assurance that their systems manifest critical functions and properties. However, the requirements for critical database systems (i.e., integrity, security, and safety) often conflict. For example, a secure system responds to an anomaly by degrading gracefully and taking no action, whereas a safety system continues to perform its intended function.

## ARCHITECTURAL DISCUSSIONS

Figure 1 depicts a generic three-layer/schema architecture (see Elmasri and Navathe 1994). The goal of such an architecture is to separate the user applications (e.g., BM/C<sup>3</sup> applications) and the physical database. Layer I, the internal layer, describes the physical storage structure of the database and is characterized by an internal schema. Layer II, the conceptual layer, describes the database from a community of users and is characterized by the conceptual schema. Layer III, the external or view layer, compartmentalizes the database to a particular view or user and hides the remainder. Layer III is characterized by an external schema or user views.

In addition, Figure 1 depicts the separation properties required for a multilevel secure database application (such as the BM/C<sup>3</sup> element of a ballistic missile defense system). The implication of this depiction is that multiple views exist, each associated with a classification. The security policy states that there shall be no (a finite, small probability of) exchange of information from a higher classification to a lower (no write down) and that a lower classification may not read information from a higher level.

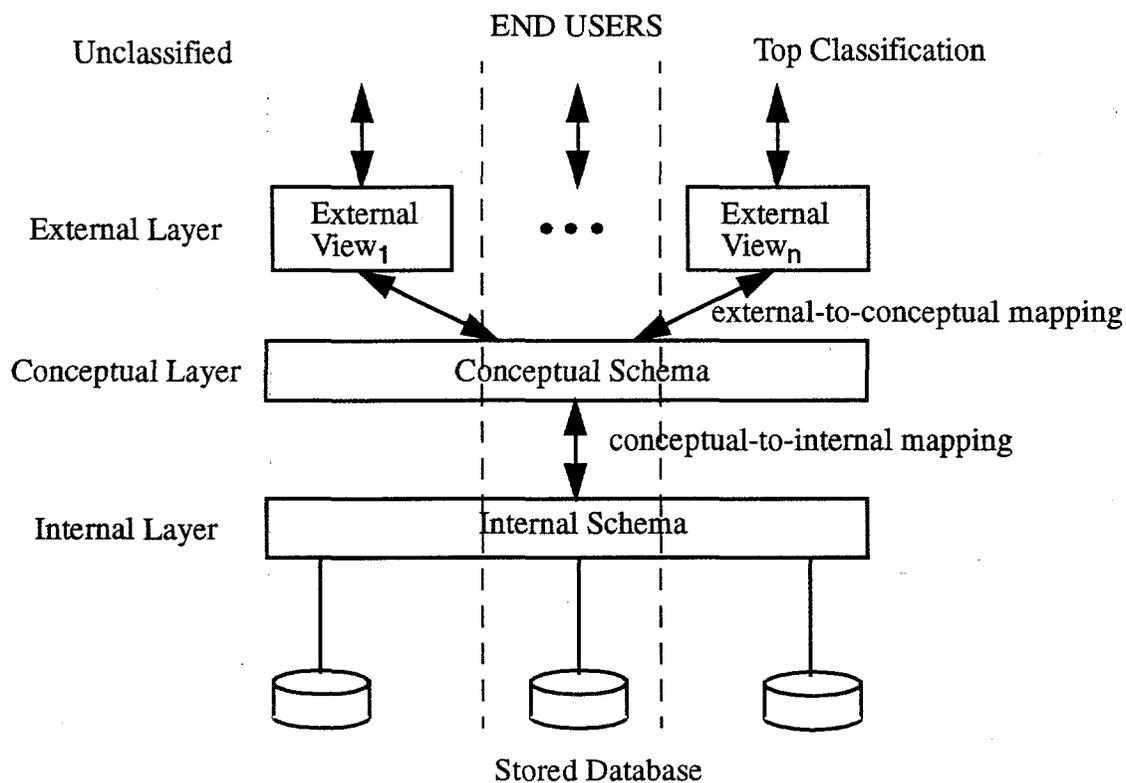


FIGURE 1. The Three-Layer/Schema Architecture

A request specified at the external layer is transformed or mapped into a request on the conceptual layer/schema. The conceptual request is then transformed/mapped for processing over the internal layer. The response at the internal layer goes through the reverse of this process to provide a response to the user's external view.

Thus a function (e.g., create) exists as a thread from the outer layer into the inner layer. This thread represents the functional devolution from the external layer in toward the inner layer. That is, for a user to create an entity or object, each layer must interface with the successive layers to effect creation at the lowest layer. To satisfy the requirements for multilevel security, an enforcement mechanism becomes part of each schema to ensure separation of data. The critical functions are composed of the innermost layer and these enforcement mechanisms.

Specifically, design criteria (e.g., security, safety, or integrity) permeate all functions and layers of the architecture. That is, to ensure that one compartment or view is separate from all others, mechanisms must be built into each schema/layer to enforce separation. Certification of these mechanisms must ensure that security, safety, and integrity are maintained throughout the design structure.

Certification and the analysis associated with the certification/accreditation process is complicated

by the requirement for a distributed capability. The remainder of this paper will strive to demonstrate the composition of critical design and functional requirements in a common model to facilitate certification of a critical, distributed database.

## 2. MODEL FOR COMPOSITION OF CRITICAL PROPERTIES

Figure 2 depicts a generic architecture for our example of a critical systems design (i.e., the BM/C<sup>3</sup> element of a ballistic missile defense system). This figure is commonly referred to as an "onion" diagram because, as components fail, the layers are peeled off, and the system gracefully degrades from full functionality to less capability while maintaining the ability to provide critical functionality.

For example, Figure 2 depicts a generic architecture for the BM/C<sup>3</sup> database system example. The intent of this diagram is to show layering and data separation/compartmentalization (e.g, sensing, planning, and execution data). The concentric rings represent the layers described for the three-layer/schema architecture, and the inner circle depicts the critical computer/network resources required to support a multilevel, distributed system. Such a system depends on a trusted computer base at each location and on secure channels between locations for data transfer. The following discussion uses the secure channel as the basis for describing an architectural model that permits composition of critical functional and design requirements in a framework amenable to the certification process.

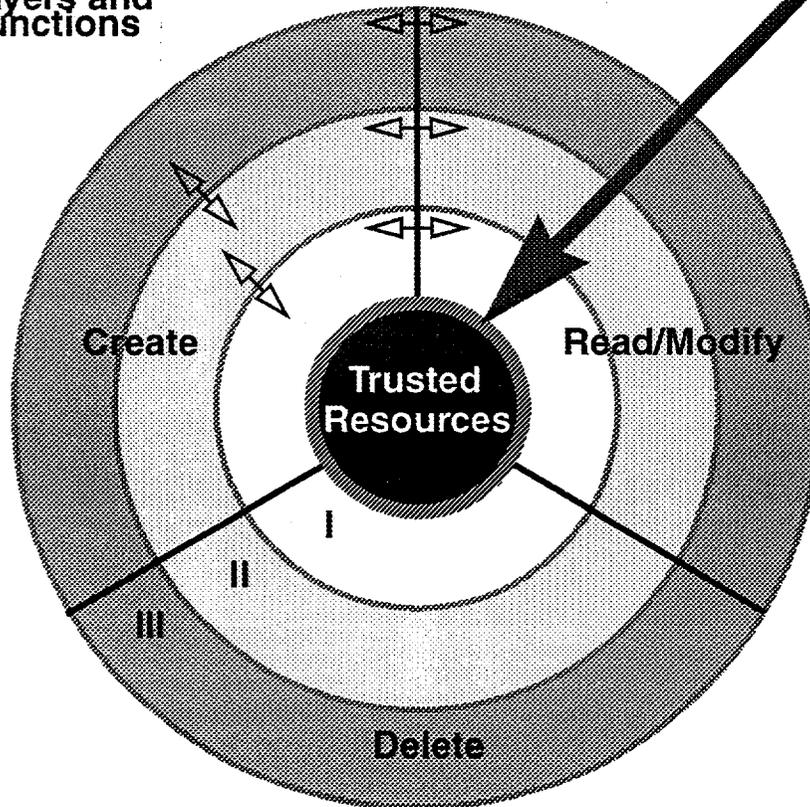
The design criteria for such a system are to:

- Maintain security (previously defined as "no write up, no read down" between security levels/views),
- Perform critical functions for the BM/C<sup>3</sup> (i.e., destroy only valid threats by *sensing* only valid threats, *planning* to target only valid threats, and *execute targeting* only on valid threats), and
- Perform the critical functions and remain secure in the presence of any single point of failure (a definition of the safety property).

To illustrate the proposed model in the context of the generic architecture, we focus on the message passing subsystem (a subsystem of the trusted computer resources). This subsystem is essential to establishing secure channels between the distributed subelements of the BM/C<sup>3</sup> system. Thus the message passing subsystem embodies critical functionality while also maintaining security and safety properties.

↔ A Membrane provides separation between layers and between functions

Enabling Critical Resources, e.g., msg. passing



### Three Layer/Schema Architecture

- I. Internal Layer - Internal Schema
- II. Conceptual Layer - Conceptual Schema
- III. External Layer - External Schema/User Views

### Essential BM/C<sup>3</sup> Functional Requirements & Data Separation:

- A. Sense
- B. Plan
- C. Execute

### Database Functions

- i. Create
- ii. Read/Modify
- iii. Delete

FIGURE 2. Generic Architecture for a Safe, Secure, Distributed Database

Our approach is based on Hoare's theory [Hoare 1969] as described in the following outline:

- Build a decomposition tree for the functional system (we use flow net examples),
- Formally show that each level of the tree is an implementation of its parent in our formal system [Chisholm 1997], and
- Show that the final layer is safe on the basis of a derived safety theorem.

We assert that a comparable demonstration for security (for the final layer) ensures a safe, secure, distributed database system that embraces the integrity requirements and will withstand the certification/accreditation process.

Our design paradigm is central to the two concepts for complexity reduction: a function is decomposable and layered. Decomposition refers to the subdivision of a function into multiple subfunctions until a least-reducible element is attained. A function (subfunction) establishes a layer, with separation between each layer in accordance with a requirement to contain failures (a safety requirement). The innermost layer is composed of the critical elements of a function. The trusted computer resources support the database system and must be evaluated more rigorously than the outer layers. We concentrate on one example of these resources, the message handling subsystem (see Appendix A for more detail).

The development of the following model depends on two activities: developing flow net specifications and reasoning about these specifications. Flow nets are engineering tools that assist developers in visualizing the control and data flow of a design. Reasoning about flow nets involves developing a clausal representation of the specification and computer-assisted analysis by using an automated theorem-proving program developed by Argonne (OTTER). The analysis component described here is based on Hoare's (Hoare 1969; Floyd 1967) partial correctness model (extended for flow net systems) and the formal semantics of the strongest postcondition (Gannod and Cheng 1995; Dijkstra and Scholten 1990; Gries 1981).

For our example of the message passing subsystem, we wish to demonstrate that we have a "safe" implementation. A safe implementation is described as one that accomplishes the functions of send, transport, and receive in the presence of a single failure. The critical functions of the system (e.g., create, read/modify, and delete for each of the three data compartments: sense, plan, and execute) must comply with the critical properties of the system (i.e., security and safety). This implies that the critical functions must operate while the system is secure at all levels (no write up, no read down) and performs the critical, secure functions in the presence of a single failure (the safety property). We describe this in the context of a critical resource (i.e., the messaging subsystem). Messaging failures may be characterized as a "false," "corrupted," or "lost" message in the formal model, as conceptually outlined in the appendix. Briefly, a false message is a message that is in the system, but has no origin. A corrupted message is one that has been altered by the system and is no longer valid. Finally, a message may be lost by the system. The appendix is viewed as a specification for a system that must ensure correct operation of the message subsystem and provide the formal machinery for mapping from this conceptual description down to the implementation details (only the conceptual details are contained in the appendix).

### 3. SUMMARY

The database design requires multiple levels of security (i.e., top secret, secret, confidential, and unclassified). Data associated with these levels will be separated on the basis of the following paradigm: *nothing is written from a more secure level to a less secure level, and nothing is read from any level by a less secure level (no write up, no read down)*. Such a paradigm may be superimposed on Figure 2, where the innermost layer is the most secure and security requirements lessen toward the outer layers.

We propose a paradigm in which the functions of the database are layered and critical properties (e.g., safety, security, and integrity) permeate all layers equally. Such a paradigm provides for a reduction in the complexity of critical functions and an associated reduction in the analytical burden associated with accreditation. We describe an approach to design that demonstrably reduces the scope for ensuing analysis in support of accreditation. This approach is based on a formal mapping of requirements from the conceptual design to the implementation, with a rigid proof that the system when implemented performs the critical functions while preserving critical properties. We illustrate this paradigm with a description of a message passing subsystem that is essential to the functionality of the example system. We assert that composition of critical properties is based on first demonstrating that the design implements the requirements (integrity) by using a formalism based on Hoare's strongest postcondition. Subsequently, a theorem specifying the safety properties is proven over the implementation. An additional theorem specifying security properties may be proven over the implementation and composed with the safety and integrity proofs.

### 4. BIBLIOGRAPHY

- Chisholm, G. H. (1997). *Toward the composition of safety and security properties - case study of a prototypical cryptographic subsystem*. Technical Report ANL/DIS/TM-34, Argonne National Laboratory.
- Dijkstra, E. W. and Scholten, C. S. (1990). *Predicate Calculus and Program Semantics*. Springer-Verlag, New York.
- Elmasri, R. and Navathe, S. B. (1994). *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, second edition.
- Floyd, R. W. (1967). *Assigning meanings to programs*. In Schwartz, J. T., editor, *Mathematical Aspects of Computer Science*, pages 19-32. American Mathematical Society, Providence, R.I.
- Gannod, G. C. and Cheng, B. H. C. (1995). *Strongest postcondition semantics as the formal basis for reverse engineering*. In *Proceedings of the IEEE Working Conference on Reverse Engineering*, Toronto, Ontario.
- Gries, D. (1981). *The Science of Programming*. Springer-Verlag, New York.
- Hoare, C. A. R. (1969). *An axiomatic basis for computer programming*. *Communications of the ACM*, 12:576-580.

## APPENDIX A: MESSAGING SUBSYSTEM MODEL

This appendix describes our model for a message passing subsystem, which is a critical component of the infrastructure supporting the three subelements of the BM/C<sup>3</sup> database system. We describe this model at a level of detail that abstracts the functions for an unstructured network. Each subelement site interfaces to the network through two buffers: a send buffer and a receive buffer. A site places messages into its send buffer when they are ready for transmission to another location. The network transports the messages to the destination, where they are placed in that location's receive buffer. Details of the movement of messages are unspecified at this level.

The following material describes a generic flow net specification, a flow net specification for a message subsystem, and reasoning about the specification for the message subsystem.

### A.1 A GENERIC FLOW NET

Figure 3 depicts a generic flow net, consisting of three states (First\_state, Second\_state, and Third\_state) and two transitions (First\_transition and Second\_transition).

Associated with the First\_state are two lists: [Precondition\_structure, precondition\_values] and [Initial\_state, initial\_values]. These lists emulate the first element in a Hoare triple, namely, the precondition.

Associated with the First\_transition is an assignment statement, M\_Asn(NOP, don\_t\_care). The interpretation of this statement is as follows: M\_Asn depicts an assignment statement where 0 or more variables may be assigned new values. Additionally, this statement computes a postcondition, based on the semantics of the assignment statement. (See the discussion on strongest postcondition for a more formal treatment.)

The postcondition for First\_transition is represented in a flow net as a token. Thus, the postcondition of one transition becomes the precondition of the subsequent transition. Again, the semantics are tied to the Hoare theory. The Second\_transition is similar to the first, except that the assignment becomes one of multiple assignments to multiple variables: Ident\_var\_1 and Ident\_var\_2. Note that the notation for a variable consists of a tuple, the first element being the identifier and the second the assigned value.

The two lists represent pre/postconditions and local variables. The assignment function differs for each list. For the pre/postcondition list, the values are appended to the list. This represents the semantics of the assignment axiom (Hoare 1969). The treatment of the second list allows the values associated with an identifier to be altered. Thus, the lists associated with the Second\_state show the effect of No Operation (NOP), and the lists associated with the Third\_state show the effects of assigning values to two variables: specifically, the values are appended to the first list and inserted into the second (if the values had previously been assigned, they would have changed).

Flow nets represent functionality and correspond to the functional requirements in a specification. Therefore, each transition can be further defined. Usually, further definition of transitions continues until a least-reducible element is attained. In the following example, we present a flow net specification: at a conceptual level, further define the transitions in separate flow nets, and present a final

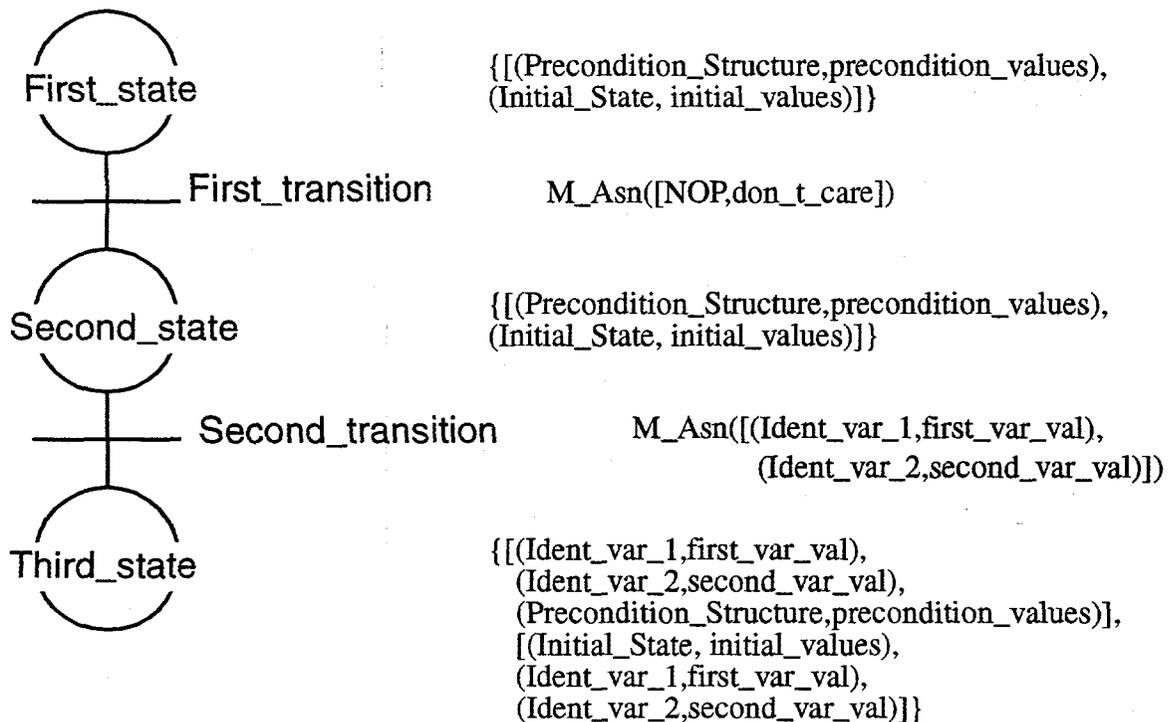


FIGURE 3. Generic Flow Net

representation that shows a low-level implementation as well as specific theorems associated with safety properties.

## A.II FLOW NET SPECIFICATION OF A MESSAGE SUBSYSTEM

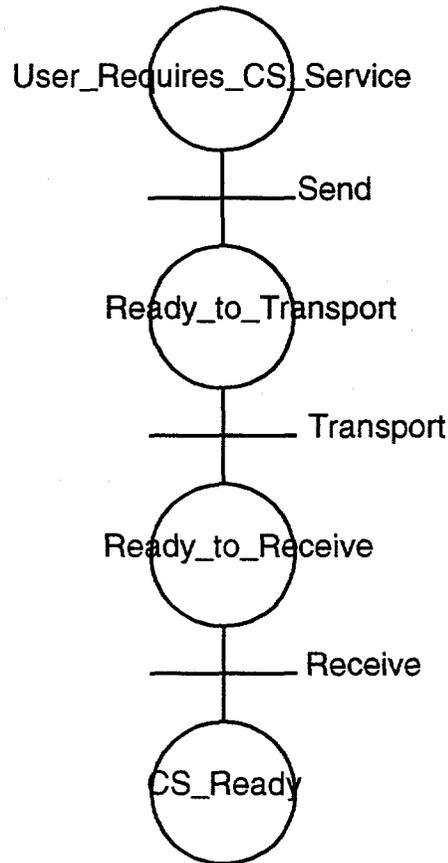
Figure 4 depicts an intermediate specification for a messaging subsystem. The specification process consists of instantiating the generic file to details specific to the system under consideration. Initially, a conceptual design for a system is described (e.g., the three layer/schema architecture). The corresponding diagram is translated into a clausal representation and submitted to the OTTER automated theorem-proving program for processing. When a proof is received, the conceptual functions are decomposed into subfunctions. These subfunctions are then encoded in the language of OTTER and reasoned about in the context of the Hoare formalism (see Chisholm 1997 for examples). The process continues until the design is complete. At this stage, we claim that the functional hierarchy is complete. The next phase in the process is to show that the functional implementation possesses the critical properties (e.g., safety and security; integrity is manifest in the formally derived hierarchy).

## A.III REASONING ABOUT THE MESSAGE PASSING SPECIFICATION

For our example of the message passing subsystem, we wish to demonstrate that we have a "safe" implementation. A safe implementation is described as one that accomplishes the functions of

Precondition:

{Message\_User\_Process\_Id,identifier,  
Message\_Destination,destination\_iden-  
tifier  
Message\_Data,clear\_text\_buffer  
Message\_protection\_mechanisms,  
set\_protection\_mechs



Postcondition:

{Message\_User\_Process\_Id,identifier,  
Message\_Destination,destination\_identifier  
Message\_Data,clear\_text\_buffer  
Message\_protection\_mechanisms,  
set\_protection\_mechs}

FIGURE 4. Second-Level: Message Subsystem

send, transport, and receive in the presence of a single failure. Messaging failures may be characterized as a "false," "corrupted," or "lost" message.

The model of the messaging subsystem is presented in first-order predicate logic. An operating system environment contains messages, threads, memory, and ports. Located in this environment are senders, transporters, and receivers of messages (threads, memory, and ports are beyond the scope

of this model, and type enforcement has been omitted for simplicity). Objects are sent, transported, and received. An object is in a send\_buffer, receive\_buffer, or the network. The environment is discrete and can be defined as a set of "objects."

- $\text{Object}(x)$  -  $x$  is an object.
- $\text{Message}(x)$  -  $x$  is a message.
- $\text{Send\_Buffer}(x, \text{Send}(y, z))$  - Entity  $y$  is sending a message  $x$ , to  $z$ .
- $\text{Network}(x, \text{Transport}(y, z))$  - Message  $x$  is being transported  $y \rightarrow z$ .
- $\text{Receive\_Buffer}(x, \text{Receive}(y, z))$  - Message  $x$ , sent by  $y$ , is received by  $z$
- $\text{LM}(x, F(y, z))$  - "Lost Message."  $x$  is a message and it has been lost.
- $\text{FM}(x, F(y, z))$  - "False Message."  $x$  is not a message but is in the Send\_Buffer, Receive\_Buffer, or network.
- $\text{CM}(x, F(y, z))$  - "Corrupted Message."  $x$  is a message that has been corrupted.
- $\text{Correct}(x, F(y, z))$  - Given locations  $y$  and  $z$ , the message is correct.
- $\text{T\_r}(x)$  -  $x$  is a valid message.
- $\neg \text{T\_r}(x)$  -  $x$  is not a valid message.
- $\text{T\_e}(x, F(y, z))$  - Given input  $y$  and  $z$ , to the O/S,  $x$  is a message.
- $\neg \text{T\_e}(x, F(y, z))$  - Given input  $y$  and  $z$ , to the O/S  $x$ , is not a message.
- $\text{Message\_Sent}(x, \text{Send}(y, z))$  -  $x$  is a message, sent from  $y$  to  $z$ .
- $\text{Message\_Received}(x, \text{Receive}(y, z))$  -  $x$  is a message, received at  $z$  from  $y$ .
- $\text{Message\_Transport}(x, \text{Network}(y, z))$  -  $x$  is a message, sent from  $y$  to  $z$  and currently being transported by the network.

A "false message" is an object that exists in either the Send\_Buffer, Receive\_Buffer, or network but is not a valid message.

Chisholm 1997 provides a detailed description of this model and its manipulation via the OTTER automated theorem-proving program.

#### A.IV SUMMARY

This appendix provides a brief introduction to flow nets, a flow net description for a message passing subsystem, and an example of the manipulation of flow net specifications by using the OTTER automated theorem-proving software. This discussion is presented in the context of using such mechanisms for the specification and verification of critical system properties. The effort associated with such analysis is time-consuming, and design paradigms that reduce complexity, (i.e., as depicted by the "onion" diagram) are essential for such analyses.