[3] A. Borning, R. Duisberg, B. Freeman-Benson, A. Kramer and M. Woolf. Constraint Hierarchies. In *OOPSLA '87*, 48-59, Orlando FL, October 1987.

[4] R. Gotzhein. Knowledge-Oriented Consideration of Communication Protols. In IFIP WG 6.1 Eighth International Symposium on *Protocol Specification, Testing, and Verification*, 295-306, Atlantic City NJ, June 1988.

[5] V. Hadzilacos. A Knowledge Theoretic Analysis of Atomic Commitment Protocols. In Sixth *Principle of Database Systems*, 129-134, San Diego CA, March 1987.

[6] J. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In Third *Principle of Distributed Computing*, 50-61, Vancouver BC, Canada, August 1984.

[7] J. Halpern and R. Fagin. Modelling Knowledge and Action in Distributed Systems. *Distributed Computing* (1989) 3:159-177.

[8] G.E. Kaiser, and D. Garlan. Melding Data Flow and Object-Oriented Programming. In *OOPSLA '87*, 254-267, Orlando FL, October 1987.

[9] G.E. Kaiser and B. Hailpern. An Object Model for Shared Data. In *International Conference on Computer Languages*, 136-144, New Orleans LA, March 1990.

[10] S.M. Klerer. The OSI Management Architecture: an Overview. IEEE *Network*, 2(2):20-29, March 1988.

[11] A.A. Lazar. Object-Oriented Modeling of the Architecture of Integrated Networks. Technical Report 167-90-04, *Center for Telecommunications Research*, Columbia University, New York, January 1990.

[12] S. Mazumdar and A.A. Lazar. Knowledge-Based Monitoring of Integrated Networks. In Branislav Meandzija and Jil Westcott (editors), *IFIP TC 6/WG 6.6 Symposium on Integrated Network Management*, 235-243, North-Holland, Boston MA, May 1989.

[13] W.T. Tsai, C.V. Ramamoorthy, W.K. Tsai and O. Nishiguchi. An Adaptive Hierarchical Routing Protocol. IEEE Transactions on *Computers*, 38(8):1059-1075, August 1988.

[14] J. Tucker. A Common Approach to Managed Objects. In Branislav Meandzija and Jil Westcott (editors), *IFIP TC 6/WG 6.6 Symposium on Integrated Network Management*, 159-165, North-Holland, Boston MA, May 1989.

A.2 Consistency Constraints

Definition A.5 (Comparing Constraints) *$C^{[p_1,\epsilon_1]}$ is stronger than $C^{[p_2,\epsilon_2]}$, denoted by $C^{[p_2,\epsilon_2]} p \trianglelefteq C^{[p_1,\epsilon_1]}$, if and only if $(p_1 \geq p_2) \wedge (\epsilon_1 \leq \epsilon_2)$.*

Fact A.1

$$C^{[p,\Diamond]} \trianglelefteq C^{[p,\epsilon]} \trianglelefteq C^{[p,0]}$$

Fact A.2 (Combining Constraints)

$$(R_1, S, T, C^{[p_1,\epsilon_1]}) \wedge (R_2, T, D, C^{[p_2,\epsilon_2]}) \Rightarrow (R_{12}, S, D, C^{[p_{12},\epsilon_{12}]}),$$

where $p_{12} = p_1 \times p_2$ and $\epsilon_{12} = \epsilon_1 + \epsilon_2$.

Fact A.3 (Merging Constraints) *If R_1 and R_2 are not in conflict with each other,*

$$(R_1, S, D, C^{[p_1,\epsilon_1]}) \wedge (R_2, S, D, C^{[p_2,\epsilon_2]}) \Rightarrow (R_1 \cup R_2, S, D, C^{[p_{12},\epsilon_{12}]}),$$

where $p_{12} = p_1 + p_2 - (p_1 \times p_2)$ and $\epsilon_{12} = min\{\epsilon_1, \epsilon_2\}$.

Definition A.6 (Consistency Climbing) *A climbing from $(R, S, D, C^{[p_1,\epsilon_1]})$ to $(R, S, D, C^{[p_2,\epsilon_2]})$ with a $C^{[p_d,\epsilon_d]}$ diffusion scheme is denoted by*

$$(R, S, D, C^{[p_1,\epsilon_1]}) \vdash_{[p_d,\epsilon_d]} (R, S, D, C^{[p_2,\epsilon_2]}),$$

where $(C^{[p_1,\epsilon_1]} \trianglelefteq C^{[p_2,\epsilon_2]} \trianglelefteq C^{[p_d,\epsilon_d]})$.

Definition A.7 (Consistency Descending) *A consistency descending from $C^{[p_2,\epsilon_2]}$ to $C^{[p_1,\epsilon_1]}$ with a $C^{[p_d,\epsilon_d]}$ diffusion scheme is denoted by*

$$(R, S, D, C^{[p_2,\epsilon_2]}) \vdash_{[p_d,\epsilon_d]} (R, S, D, C^{[p_1,\epsilon_1]}),$$

where $(C^{[p_1,\epsilon_1]} \trianglelefteq C^{[p_2,\epsilon_2]})$.

References

[1] J. Bennett, J. Carter and W. Zwaenepoel. Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence. In Second *Principles & Practice of Parallel Programming*, 168-176, Seattle WA, March 1990.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs NJ, 1987.

1. Merging and combining *nested consistency constraints* for a hierarchical system design.

2. The relations between consistency climbing protocols and other consistency protocols, and the properties of a *climbing gap*.

3. Consistency analysis of various routing protocols.

4. Distributed reasoning based on the information from preconditions, consistency constraints, climbing protocols, and consistency hierarchies.

Acknowledgments

Discussions with Subrata Mazumdar have triggered most of the ideas in this paper. We would like to thank Aurel Lazar, who addressed the problem of information inconsistency in his *Network Management* course. Wenwey Hseush and Naser Barghouti gave valuable comments on an early version of this paper. Rany Chacar and Dave Staub are currently implementing an experimental network management system in the *Meld* concurrent object-oriented language.

A Common Knowledge and Consistency Constraints

A.1 Common Knowledge

Definition A.1 (Likely-ϵ-Common Knowledge) *A constraint is a likely-ϵ-consistency constraint, $(R, S, D, C^{[p,\epsilon]})$, if, with probability $0 \leq p \leq 1$, it doesn't take more than a finite ϵ time to maintain the constraint between S and D.*

Definition A.2 (ϵ-Common Knowledge) *A constraint is a 1-ϵ-consistency constraint, $(R, S, D, C^{[1,\epsilon]})$, if, with probability one, it doesn't take more than a finite ϵ time to maintain the constraint between S and D.*

Definition A.3 (Common Knowledge) *A 1-ϵ-consistency constraint is a perfectly consistency constraint, $(R, S, D, C^{[1,0]})$, if $\epsilon = 0$.*

Definition A.4 ($\Diamond$-Common Knowledge) *A 1-ϵ-consistency constraint is an eventually consistency constraint, $(R, S, D, C^{[1,\Diamond]})$, if $\epsilon = \infty$.*

4 Remarks

Consistency constraints and consistency hierarchies are proposed to specify the complicated relations among network objects, sensor objects and manager objects. A consistency constraint will not only specify the relation between two objects (or two sets of objects in general) but also declare the time and probability to maintain the relation. A consistency hierarchy, on the other hand, is like a gateway between consistency constraints and management tools. A consistency hierarchy will organize several existing management tools into a conceptual structure, which can be easily retrieved by other management tools because of the power of abstraction.

We have found several potential advantages for developing management software using both consistency constraints and objects:

flexibility: The system would become more flexible because the design goal, *maintaining consistency*, has been directly presented as a piece of data in the program, and the goal can dynamically change at run time. This will be especially helpful in designing a performance and/or fault management system, where the managers must be able to reconfigure the system at any arbitrary time.

reusability: Since nearly all environment-dependent information could be hidden, the system specification is very portable. For example, we might have two totally different ways to achieve the same consistency level for token-ring networks and bus networks.

robustness: If a single information entity is contained by two different consistency constraints, i.e. the intersection of two consistency groups, the problem of running two protocols simultaneously could be handled implicitly by the system based on the consistency hierarchy. For example, the routing table of a *gates border node* [13] in a hierarchical routing protocol may be contained in two different consistency constraints: one with lower level nodes and another with other border nodes.

extensibility: Because the specification will be highly abstract, it will be more understandable for a new designer to extend. Theoretically, if he understands both the class hierarchy and the consistency hierarchy, he will be able to design a system without the knowledge about, for example, "what the routing protocol(or protocols) are in the network."

Four interesting problems are currently under our consideration:

Figure 13: Consistency Constraints in the Example: II

Figure 12: Consistency Constraints in the Example: I

3.3 An MIB with Consistency Constraints

In order to maintain the *active* relations among network entities, sensors, and network managers, several constraints are introduced in our class definitions. However, those constraints present the relations but not the time and the probability to maintain them. For example, in sensor objects (Figure 7), the *unusual throughput value* constraint should be maintained in two micro seconds with probability one, and the normal reporting constraint can be handled in maybe ten micro seconds with probability 0.5. In some situations, even the same constraint can be manipulated in different ways. For instance, in Figure 5, although both S_a and S_b are M_1's sensors, the relation between M_1 and S_a will be closer because the network Net_a is more interesting to M_1.

With consistency specification, two constraints in sensor objects are redefined (Figure 10). The first part of a consistency constraint is the declaration of preconditions. For example, $R_{normalReport}$ has no precondition because the time interval limitation has been defined in the consistency index. Thus, $R_{normalReport}$ is an always-active constraint. For $R_{unusualT}$, the precondition is "throughput.value > normalValue", which indicates that if it is true, the throughput data will be transmitted to the parameter of the _throughput_report operation of the manager object in two microseconds with probability one. Moreover, one precondition can be shared by several consistency constraints. For instance, S_b is employed by both M_1 and M_2 and the consistency index between S_b and M_2 is stronger than the one between S_b and M_1. Therefore, S_b will behave differently under the same precondition (Figure 11).

The second part of a constraint defines a relation between a sensor and a manager with a consistency index $[p, \epsilon]$. In general, we can put two variables in a consistency index, which would allow us to change the time limit and probability at run time. For instance, in Figure 12, the consistency index from M_1 to S_b is $[p_1, \epsilon_1]$, and the one from M_2 to M_1 is $[p_2, \epsilon_2]$. Occasionally, the connection between M_2 and S_b is broken and M_2 can no longer get real-time information about Net_b through S_b. However, M_2 knows that the connection between M_1 and S_b is still working and in order to make a correct decision, it can ask M_1 for more statistical information about Net_b. M_1, after understanding M_2's situation, has to modify the consistency relation with S_b because information from S_b becomes more important now. The new consistency relations are depicted in Figure 13.

```
CLASS Manager IS Object ::=
        ATTRIBUTES:
                String    name;
                SET OF Manager    otherManagers;
                SET OF Sensor    sensorSet;
        CONSTRAINTS:
        OPERATIONS:
                _throughput_report (throughput)
                Throughput throughput;
                {
                        $SELF._make_decision("throughput", throughput);
                }
END OF CLASS Manager

CLASS PerformanceManager IS Manager ::=
        ATTRIBUTES:
                Statistics    statisticalInfo;
        CONSTRAINTS:
        OPERATIONS:
END OF CLASS PerformanceManager
```

Figure 8: A Class Definition for Manager Objects

```
CLASS Statistics IS Information ::=
        ATTRIBUTES:
                Throughput    averageThroughput, rangeOfThroughput;
                Real    averagePacketLoss;
                Time    averageTimeDelay, rangeOfTimeDelay;
        CONSTRAINTS:
        OPERATIONS:
END OF CLASS Statistics
```

Figure 9: A Class Definition for Statistical Information Objects

```
CLASS Sensor IS Object ::=
        ATTRIBUTES:
                String    name;
                Manager    manager;
                Network_Entity    netEntity;
                Time    previous, timeLimit, interval, timeDelay;
                Throughput    throughput;
                Real    packetLoss, normalValue;
        CONSTRAINTS:
                IF (($Clock._real_time () - previous) > interval)
                THEN
                {
                        SEND _throughput_report (throughput) TO manager;
                        previous := $Clock._real_time ();
                }
                IF ((throughput.value > normalValue)
                THEN
                        SEND _throughput_report (throughput) TO manager;
        OPERATIONS:
                _ne_info (infoIndex, netEntity)
                Integer infoIndex;
                Network_Entity netEntity;
                {
                        IF (infoIndex == 1)
                        THEN throughput = netEntity;
                }
END OF CLASS Sensor
```

Figure 7: A Class Definition for Sensor Objects

```
CLASS Network_Entity IS Object ::=
        ATTRIBUTES:
                String    name;
                Layer    inLayer;
                SET OF Sensor    sensorSet;
        CONSTRAINTS:
        OPERATIONS:
END OF CLASS Network_Entity

CLASS Buffer IS Network_Entity ::=
        ATTRIBUTES:
                Throughput    throughput;
                Real    packetLoss;
                Time    timeDelay;
        CONSTRAINTS:
                SEND _ne_info(1, throughput) TO sensorSet;
                SEND _ne_info(2, packetLoss) TO sensorSet;
                SEND _ne_info(3, timeDelay) TO sensorSet;
        OPERATIONS:
END OF CLASS Buffer
```

Figure 6: A Class Definition for Network Objects

M_1 will have to change its strategy in getting information. One obvious alternative way is to put all the queries through S_b and to deduce the statistical information by itself. Then, since M_1 will totally rely on S_b for Net_b, sensor S_b also has to change its behavior in terms of consistency level in order to supply sufficient information for M_1.

3.2 An MIB with Classes of Objects

Network objects are the source of dynamic real-time information, which includes the traffic load of a link, the number of packets in a buffer, the configuration information of a special node, and other information that managers are interested in. They can be implemented with some message passing protocols, which allow them to offer dynamic information for either *sensors* or *managers*. For one possible class definition (Figure 6), a *buffer*, which inherits from a network entity, contains values for *throughput*, *packet loss*, and *time delay*. With three *constraints*, [3] all these values are active because any change in them will be reported to those monitoring sensors. A set of sensors' addresses is kept in the attribute *sensorSet*, which will be set up by a configuration management protocol between sensors and networks.

Sensors objects are slightly more complicated because they will not only send mail to managers but also worry about whether this specific piece of information should be delivered or not. Normally, a sensor will report to its manager periodically, for example, every five micro seconds. However, after receiving information, a sensor should send a message to the manager if the information indicates some special situations about the network. The first constraint (Figure 7) presents normal reporting, and the second one present one special case, *unusual throughput value*.

Manager objects are the most complex because they will worry about not only how to process the received information but also how much information is needed and what the best way is to get the information. For example, a performance manager might want to set up two sensors for collecting the real-time information and one communication channel with another performance manager for getting statistical information. Therefore, a manager as defined in Figure 8 must know the addresses of its own sensors and of other managers. Furthermore, it has to keep its own statistical information for requests from other managers, *e.g.*, *average throughput*, *average packet loss*, and *average time delay* (Figure 9).

[3] The notion of *constraints* can be found in many object-oriented systems, for example, *Meld* [8] and *ThingLab* [3]. Unlike *passive operations*, which would only be triggered by external messages, constraints are active and monitor the data in an object automatically.

Figure 5: Managers, Sensors and Networks

Figure 4: Managers and Networks

of the network for *real-time information* and they will also communicate with each other for *statistical information* [11] [12]. In practice, in order to sample the information from the real network, a manager might employ one or more agents, *sensors*, to collect all the real-time information, to report periodically and to give warning under certain emergent situations. After introducing these sensor objects, we get another picture of the system (Figure 5).

S_a is much closer to M_1 than S_b is, and the information from S_a is usually cheaper and more important to M_1. So, traffic between M_1 and S_a would be very frequent and thus maintain a high degree of consistency for the information among M_1, S_a, and the sub-network Net_a monitored by S_a. However, if M_1 is interested in the subnetwork Net_b monitored by S_b, it could either directly query S_b or query some statistical information from M_2. It is M_1's task to decide how much information about Net_b is needed through each path.

This small management system will become more complicated if every object in the system has dynamic behaviors. First, the network itself is dynamic. For example, links can go down and traffic load changes frequently. Second, the behavior of a manager can change. For instance, if M_2 goes down or no longer supports correct statistical information,

Figure 3: A Consistency Gap

and it is necessary to have another *climbing* protocol for diffusing the information about the new consistency index. In other words, the consistency index itself should be *common knowledge* between source and destination objects. Moreover, if the consistency index of the diffusion protocol itself is weaker than the upgraded consistency index in probability, some objects might start supporting the new consistency index while other objects never receive the upgrading control messages. Also, a *gap* (Figure 3) will exist between the starting time of the climbing protocol and the time the new consistency index becomes *real* common knowledge [2]. The consistency in the gap is unpredictably between the old and the new index, and the length of the gap depends on the consistency index of the climbing protocol.

3 Information Model with Consistency Constraints

3.1 Example of Managers, Sensors and Monitored Networks

Consider a small network management system, which includes one real network (or a set of managed objects) and two managers (Figure 4). Both managers would monitor part

[2] If each node is running an *eager* protocol [6], it will treat the received message as *common knowledge* even if it knows other nodes might not have received the message yet.

Figure 2: A Consistency Hierarchy

2.2 Consistency Hierarchy

The definition of consistency index implies that the number of indexes is infinite and continuous. However, in practice, a finite number of indexes are adequate to present the consistency constraints in most application domains. The major reason is that each consistency index needs at least one protocol to support it, and since the number of protocols is finite [1] and theoretically consistency indexes for these protocols can be analysed (for example, [5] [4] [1]), we could use these existing protocols to partition the whole consistency space into a finite quotient set. After the partition, we get a *consistency hierarchy*, which is a acyclic directed graph (Figure 2) based on the *comparing rule* (see Appendix).

Definition 2.1 (Consistency Hierarchy) *A Consistency Hierarchy,* **CH**, *contains a finite number of consistency indexes, and* $\forall C^{[p,\epsilon]} \in \mathbf{CH}$, *there exists at least one supporting protocol.*

In certain situations, the consistency index of a constraint might change dynamically. A constraint might want to upgrade or degrade its consistency index for performance reasons

[1] It has been proven that some consistency indexes are not attainable under certain assumptions. For example, in [6], if communication is not guaranteed or clocks are not initially identical, then there is no protocol for attaining common knowledge, $C^{[1,0]}$.

Figure 1: A *LoadInfo* Constraint between a Manager and a Link

2 Consistency Constraints

2.1 Definition of Consistency Constraint and Index

A *constraint* R exists between two objects S (source) and D (destination) if a change in value of S will cause a change of D. For example, one constraint between a manager M_1 and a network link object L_1 is *LoadInfo*. If the traffic load of $L_1.load$ changes, then based on the *LoadInfo* constraint, M_1 should *know eventually* the real traffic load of L_1 (Figure 1). Since both the propagation delay and the lost message rate are unknown, possibly by the time M_1 knows the traffic load of L_1, $L1.load$ has changed to another value. In order to model a constraint's behavior completely, we introduce another factor, *consistency index*, which describes the requirement of maintaining a constraint. Therefore, we can define a consistency constraint as a 4-tuple $(R, S, D, C^{[p,\epsilon]})$, where R is a constraint (or rule) stating how S and D are related, S is the source object for R, D is the destination object for R, and $C^{[p,\epsilon]}$ is a consistency index presenting the time (ϵ) needed to maintain R and the probability (p) of correctly maintaining R.

In the manager and link example, we can state the constraint *LoadInfo* as:

$$(R_{LoadInfo}, L_1.load, M_1.loadinfo, C^{[1,d]})$$

if with probability one, the current value of $L_1.load$ must be known by M_1 in d time.

Our notion of consistency index is based on the common knowledge work by Halpern and Moss [6]. Several kinds of common knowledge can be treated as some special cases of our consistency indexes. We describe such connections in the appendix.

1 Introduction

Network management involves manipulating different kinds of information entities, *managed objects* [10] [14], which together present the status of the network. In *centralized information* environments, when a change in value of an object occurs, all objects affected by this change must also update their value immediately in order to maintain a *consistent view* for network managers. For example, if a *link* object changes its status from **connect** to **disconnect**, then the *routing information* object must update its value for the routing manager to find an optimal path. However, in a *distributed information* model [9], we usually could not force all the affected objects to update their values at the same time because of the delay of control messages. Therefore, managers in different nodes will have different views of the entire network and they might make some conflicting decisions that would severely degrade the performance of the network. For instance, when a link was disconnected, the routing manager might select a path before the update of the routing table occurs. Since the router's view of the network is out of date, the path selected might not be optimal.

Although a globally consistent view is impossible, the network is still managable with a slightly inconsistent information base in many cases. For the routing example, we will still get an optimal route if the failed link is not selected. For several robust routing algorithms, even if the routing table changes frequently, we will still be able to get a nice solution [2]. However, we might ask ourselves several interesting questions. For instance, "how do we define the term *slightly inconsistent*?" or "can we define *different levels* of slight inconsistency?" Finally, we should ask "in a particular situation, what kind of slight inconsistency is *good enough*?"

In this paper, we attempt to answer these questions. In next section, we will define what a consistency constraint is. Various kinds of consistency constraints and their relations with *common knowledge* will be described in the appendix. We also introduce the concept of *consistency hierarchy*, which presents a gateway between consistency constraints and the *management protocols* that would actually maintain the consistency between network entities. In section 3, building a network management information model using consistency constraints is considered. Finally, we discuss some possible future extensions.

Network Management with Consistently Managed Objects

Shyhtsun F. Wu [*] *Gail E. Kaiser* [†]

Columbia University
Department of Computer Science
New York, NY 10027
212-854-8182
wu@cs.columbia.edu

April 3, 1990
(CUCS-013-90)

©1990, Shyhtsun F. Wu and Gail E. Kaiser
All Rights Reserved

Abstract

A *consistency constraint* exists between two objects in a network *management information base (MIB)* if a change in value of one object will cause a change in another. Based on the definition of *common knowledge* by Halpern, Fagin, and Moses [6] [7], such constraints can be classified by the time needed to maintain them and the probability of correctly maintaining them. In this paper, we study the practical issues of how to build an MIB with a set of objects and consistency constraints. We will go through an experimental MIB example showing that distributed network entities can be consistently managed in an elegant way, even when the network environment is very dynamic.

Keywords: Information Models, Consistency Constraints, Managed Objects.

---

[*]Wu is supported in part by the Center for Telecommunications Research.

[†]Kaiser is supported by National Science Foundation grants CCR-8858029 and CCR-8802741, by grants from AT&T, Citicorp, DEC, IBM, Siemens, Sun and Xerox, by the Center for Advanced Technology and by the Center for Telecommunications Research.