# A Troubleshooting Aid for Asynchronous Data Communications Links

Schooner is an expert system for fault diagnosis and personnel training on point-to-point datacom links.

*by Brian T. Button, R. Michael Young, and Diane M. Ahart*

The data communications market is among the fastest growing markets of the computer field, and data communications support has a rapidly increasing need of expertise. Solutions in this area are perceived by computer vendors and customers to be critical for a successful total solution. The trend toward personal computers and office automation has given increasing importance to easy installation and support of data communications links.

To address a portion of this problem, Hewlett-Packard initiated the Schooner project. Schooner's principal objective is to maximize leverage of HP's expertise in asynchronous data communications and reduce the time it takes to solve datacom problems for HP's customers. It attempts to do this by solving the easier problems, thereby reducing the workload on HP's more experienced engineers and freeing them to tackle other problems. Schooner also serves as a training tool for less experienced personnel.

**Overview**

Schooner combines an inference engine and a knowledge base to provide expert-level assistance with asynchronous, point-to-point data communications problems for fault diagnosis and personnel training. It verbally guides Response Center engineers, field support personnel, or other users through the solution of problems in this area.

The present knowledge base is oriented towards diagnosis of problems in RS-232-C links connecting a terminal to an HP 3000 Computer either directly or with modems. When initiating a troubleshooting session with Schooner, the user designs the configuration by specifying any combination of terminal type, port type, operating system, connection type (cable or modem), and if applicable, the types of modems at the terminal and computer sides of the connection.

After acquiring this information, Schooner goes to the main investigation phase of the session, asking the user to relay observations and perform manipulations. Since the configuration description determines the rule set used in this phase, the session proceeds differently for modem-connected data links than for direct (cable-connected) links. Schooner understands problems and characteristics specific to the makes and models of devices in the configuration.

**Schooner Tests**

In Schooner, tests are the basic units of inference corresponding to rules in classic rule-based systems. Tests describe a state in which observations may be made, observations to make, and a translation from this information to beliefs about potential faults and the state of the system being debugged.

Although tests are the basic units of inference, there are important differences between tests and rules. Tests do not chain. Each test is a unit and is applied singly without direct interaction with other tests. Unlike a rule, once selected and applied, a test is no longer eligible for selection unless disapproved.

Under appropriate circumstances, an applied test can be disapproved. All inferences resulting from the application of the test are retracted and the test is made eligible for reaplication.

The application of a test has three stages. Each of these stages has corresponding fields, which allow the user to specify what should occur there.

- **Preliminaries.** Perform the manipulations to the system necessary before the application of the query and record assumptions made for the test.
- **Query.** Obtain the desired information.
- **Inferences.** Update beliefs about the system based on the information obtained.

**Fig. 1.** An example of a Schooner test. This test only applies to modem links. It consists of putting the terminal in a normal state to transmit, putting the modem in local loopback mode, and then typing on the terminal to see if characters come up on the screen. This test is used to verify the link between the terminal and the local modem.
Preliminaries

Two fields in a test, the precondition and requirement clauses, are provided to allow the user to manipulate the system into the state necessary to apply the query. They are both ordered lists of simple clauses, but they differ substantially in intent and use.

In the test shown in Fig. 1, the precondition clause:

?modem in-local-loopback yes

specifies a state of the modem that must be satisfied before application of the query. After the precondition clause is satisfied (Fig. 2), the modem will be in local loopback, regardless of its state before the test was selected.

Requirement clauses are used to establish an assumption about the state of the system. To satisfy a requirement clause, Schooner must bring the system to a state where it can assume that the clause is satisfied (Fig. 3). Having done so, Schooner must then tag the assumption so that the test information and application can be retracted if the assumption later turns out not to be valid.

In the test in Fig. 1, the requirement clause:

?term in-working-state yes

specifies that Schooner must put the terminal in a state in which it can be assumed that it works. If some previous test has put the terminal in local mode, then Schooner must ask the user to put the terminal in remote mode. Once this has been done, Schooner can assume that the terminal is in a working state, tag the dependency, and proceed to the next clause in the test.

Queries and Inferences

The query is used to obtain information once the system

Is there a present value in knowledge base?

Yes

Is the value assumed?

No

Ask user for value

Does the value match the desired value?

Yes

Exit

No

ask the user to establish the desired value

Exit

Fig. 2. Algorithm for satisfying a precondition clause.

is in a state appropriate to make the observation. A query is never used to achieve a state, but only to obtain information. Hence, the item in the third position in a query clause is always a variable (in Schooner, variables start with a question mark) rather than a desired value. The user is asked for the information needed by the query and the variable in the third position is set to that value. During the inferences, that value is substituted for the variable whenever it appears.

The inferences use the binding of variables set in the query to update Schooner's beliefs. Inferences may either assert beliefs about potential faults in the system or assert beliefs about the state of the system configuration.

Inference Engine

The Schooner inference engine is designed specifically for troubleshooting. It uses a general troubleshooting process (Fig. 4) to investigate potential faults in the system being debugged. Schooner applies tests to the system to determine more about faults that may exist. The result of each test increases or decreases belief in various possible faults in the system being debugged. Subsequent tests are selected on the basis of such beliefs and other information provided by previous tests.

Data Structures

The Schooner inference engine uses several partially interlinked data structures. These provide an internal representation of the data link being debugged and maintain a consistent set of beliefs about the faults therein. The most important data structures are the configuration, the faults,
the tests, and the linkage.

**Configuration.** The configuration (see Fig. 5) is a hierarchical structure that represents the system being debugged. See the box on page 46 for a discussion of hierarchies. Each node in the configuration, called an entity, is a frame representing a component of the system.

Slots in the entity frame describe aspects of the state of that component. During the specification stage each slot contains the following information: data type, possible values, actual values, default value, and cost of determining value. During the troubleshooting session, each frame also holds the current beliefs regarding that aspect of the entity as well as other pertinent information such as how, when, and where those beliefs were obtained.

In addition to the normal slots, each entity has an operating summary slot named in-working-state. This slot reflects the state of all the slots in the entity that are operating. For example, a value of yes in the operating summary slot of the data link in Fig. 5 indicates that Schooner was able to assume that the data link, the phone link, and both modems are in an operating state.

The operating summary provides a powerful capability for describing test dependencies. A test whose requirements assume that the entire configuration was in a working state doesn’t have to specify each individual entity to guarantee useful backtracking. This characteristic allows reference to the operability of the configuration at any functional level. In the example of Fig. 5, operating summary slots could be used to refer to the operability of the configuration, the data link, either modem, or in general, any entity in the configuration.

**Faults.** Faults are used for characterizing and manipulating failures that can occur in a system. They are also used for heuristic test selection (see below) and provide an explicit structure for holding current beliefs of what might be wrong in the system. During the specification stage, a fault simply contains a description and a list of tests that point to it (Fig. 6).

Faults are integrated into the configuration by their attachment to entities in the configuration. Faults are attached to the entity in which they occur (Fig. 7). For example, a fault representing a malfunction in the terminal would be attached to the terminal entity, but a fault representing a baud rate discrepancy between the terminal and the port would be attached to the configuration entity.

**Tests.** Tests are pointed to by faults. The outcome of a test can either confirm or deny a fault.

**Linkage.** The three data structures above describe the essentials necessary for troubleshooting a physical system. Schooner contains an internal description of the system being diagnosed, a set of beliefs of potential faults, and a way of describing tests to be performed on the system. The linkage data structure (Fig. 7) ties these three elements together to provide useful access when and where necessary. It provides links from an entity in the configuration to faults that may exist in the entity, and it provides links from a fault to tests that might tell about it. Once a test is applied, it finds the faults to which to assign status.

In summary, the main duties of the linkage data structure are to ensure knowledge consistency, guarantee uniqueness, and tie the whole thing together.

---

**Fig. 4.** Schooner troubleshooting flow diagram.

**Fig. 5.** Configuration data structure.

**Fig. 6.** An example of a fault in Schooner. This fault occurs when the card for the keyboard inside the terminal becomes loose in its slot.
Heuristic Test Selection

With an understanding of the essentials of the Schooner data structures, it is now possible to describe more clearly some of the other processes in the Schooner troubleshooting flow diagram, Fig. 4.

The test selection process (the box marked A in Fig. 4) is critical for rational behavior and proficient troubleshooting. Poor selection of tests will irritate the user, make troubleshooting arduous, and create an impression of irrationality. This aspect of the system can single-handedly determine the difference between acceptability and unacceptability of an expert system.

Each state change or belief change caused by a system manipulation or test completion causes Schooner to do a complete reevaluation of all possible tests to ensure an action appropriate to the new situation (Fig. 8). Each potential fault in the system is given a numeric pursuit desirability, which is a combination of its frequency (a static value assigned at specification) and its likelihood (a dynamic value received from tests during the session). Each test that can further indicate or disindicate the fault is given a numeric cost (see below). Schooner then selects the test-fault combination that best combines the attributes of low user effort (in the test) with high pursuit desirability (in the fault).

The total cost of a test is the effort to do the query plus the effort required to put the system in a state to do the test. The resulting value is a function of the effort required to perform the test from the present state of the system. This technique for determining cost tends to minimize manipulations to the system by making Schooner take maximum advantage of the present state. For example, if the user is asked to run a diagnostic to look at the state of a port, the effort required to do a system dump will cause Schooner to obtain all the information possible from doing the dump before going to another area of investigation.

Review Beliefs

When a fault is discovered and subsequently resolved in the system being diagnosed, Schooner reviews the information and consequent beliefs that have accumulated during the session. Typically, some of the information gained from previous tests and observations is no longer valid. Schooner must find the invalid beliefs and retract them. The nature of the Schooner data structures makes this process fairly simple. Schooner merely has to determine (according to procedures described below) which tests are no longer valid and "disappear" them. This procedure is what occurs in the review beliefs process (the box marked B in Fig. 4).

![Fig. 7. Linkage data structure. Faults are attached to an entity. A fault's likelihood results in a test's selection. The test manipulates the entity and infers about the fault.](image-url)

For each Fault in possible faults
Evaluate Fault pursuit desirability
For each Test that can tell about Fault
Evaluate Test Cost
Test-Desirability = Fault-Desirability/Test-Cost
End For
End For

Select Test with Maximum Desirability

Fig. 8. Schooner heuristic selection algorithm.

When a test is dispelled, all its results are retracted, that is, all status assigned to faults as a result of applying the test is retracted. Additionally, the test is made available for reapplication.

There are two reasons for Schooner to dispel a test. The first occurs when the test indicated a fault that was subsequently resolved. Since the test results were based on the existence of that fault, such actions are necessary to maintain consistency. The reapplication of the test would be very likely to yield different results, since the fault has been remedied.

The second reason for dispelling a test occurs when the test depends on an assumption specified in a requirements clause that the discovery of the fault has contradicted. In the example test shown in Fig. 1, the clause required an assumption, since it wouldn't be known that the terminal was in a working state. The subsequent discovery of a resolvable fault in the terminal (for example, being in local mode) would cause Schooner to instruct the user to resolve the fault. Schooner would then go into the review beliefs process, which would notice that the test do-local-loopback-at-modem made an invalid assumption. All beliefs that the test asserted would be retracted and it would be made eligible for reapplication.

The resulting behavior is a natural sort of backtracking and attention control. As Schooner applies tests that indicate a failure in an area of the link, faults in this area are concentrated on. If a fault is discovered and resolved, then Schooner backtracks, verifying symptoms that were discovered earlier to determine the present characteristics of the link being diagnosed and to decide where to investigate next.

These dependencies are a natural representation of the use of tests as a unit of knowledge. There are no special requirements for the specification of dependencies. All the right information gets back out when it should be.

Behavioral Results

Schooner has turned out to be a competent expert system in the domain in which it has been applied. It is effective at discovering faults for which it has knowledge. Knowledge acquisition and formalization, although cumbersome, allows Schooner more or less unlimited growth of expertise.
Hierarchies

Hierarchies are a potent tool in artificial intelligence applications. A hierarchy is an n-ary tree with a root node. Each node is represented as a frame. The frames are connected by pointers that point from parent node to child. In a hierarchy, each parent (ascendant) node is a generalization of its descendant nodes. To explain this more clearly, this article will discuss two types of hierarchy.

Inheritance Hierarchies

Inheritance hierarchies are used to describe objects in the real world. In an inheritance hierarchy (see Fig. 1), AKO (a kind of tree, higher nodes on the tree represent more general categories while lower nodes are more specific. At each level in the hierarchy, nodes contain properties that distinguish them from their siblings. Information common to all siblings is inherited from parent nodes. If the user queries a property of a node in the hierarchy, access routines (Fig. 2) search for the requested information, looking first in the specified node and then in ascendant nodes.

For example, in Fig. 1, the mammal node would hold the property of being warm-blooded. Any queries about this property to the ape, dog, or bat nodes (or nodes descendant from them) would result in the correct value being returned.

This method for representing objects in the real world is used widely in artificial intelligence. It has proved to be a very powerful representation technique for several reasons. It provides enormous economy of data; each internal representation of every kind of bird does not contain information that it breathes, eats, mates, flies, etc. This information logically, conceptually, and sensibly belongs at higher levels in the hierarchy. Adding new objects is easy. The addition of a new type of hawk called the kitty hawk would automatically result in the right sort of assumptions made about it—it flies, is carnivorous, etc. Only information that differentiates the kitty hawk from other types of hawks would actually have to be added to the kitty hawk node.

Functional Hierarchies

For troubleshooting, expert systems need sophisticated techniques for representing the systems they are debugging. Schooner and IPT (see article, page 48) represent devices they are debugging by creating an internal representation that mirrors the device's functional properties. Each device (see example in Fig. 2 on page 49) is shown as the union of its major subsystems. Each subsystem, in turn, is divided into smaller subsystems until it reaches the smallest size of component the troubleshooting system can reason about. This type of representation of a device is called a functional hierarchy.

Fig. 1. Animal characterization hierarchy.

Fig. 2. When a user queries a property of a node, access routines search for the requested information.

Functional hierarchies offer several advantages for device representation. First, they allow a divide-and-conquer troubleshooting methodology. Instead of verifying the operability of each subcomponent at the lowest level, the expert (or expert system) can determine the operability of one of the subsystems. Only when a fault has been isolated to a subsystem does the troubleshooter then investigate the nodes that lie below the subsystem.

Another advantage of functional hierarchies that is used by Schooner is that the operability of a node at any level can be viewed as the sum of the operability of all nodes descendant from it. Schooner automatically maintains this relationship, updating its beliefs of the operability of nodes at intermediate and upper levels based on the discovery of faults at lower levels. As a result, knowledge for Schooner can refer to the operability of a device with assurance of appropriate results. For example, if the electrical system in Fig. 2 on page 49 was determined to be in a working state, Schooner would merely have to assert that fact rather than eliminate all the faults in or below that node in the diagram. This presents a concise way of giving information and preserves accuracy even when faults are subsequently added or removed.
with few expected changes to the inference engine or knowledge representation technique.

The following transcript from a sample session demonstrates the knowledge base and provides a picture of Schooner's behavior with respect to the concepts described above.

Sample Session
This example demonstrates simple backtracking.
Problems: remote mode key up and terminal disconnected from cable.

WELCOME TO SCHOONER!

<<Debugger for asynchronous, point to point, direct connect, terminal problems.>>

Describe the type of problem you are experiencing.

→ inability-to-log-on

Device name for reference: PORT-3, CABLE-3 and TERMINAL-3

→ acknowledge

Using the cursor control key, move the cursor out from the left margin. Depress the return key, what appears on the screen?

→ cr-only

Is TERMINAL-3's remote mode key in the down position?

→ no

Depress TERMINAL-3's remote mode key.

→ acknowledge

Using the cursor control key, move the cursor out from the left margin. Depress the return key, what appears on the screen?

→ nothing

Schooner believed a problem had been solved. It then found out what the symptoms were and discovered a problem still existed and the symptoms are now different.

Is TERMINAL-3's block-mode key in the down position?

→ no

Hold down the ctrl key and press q.

→ acknowledge

Is any text coming up on the screen?

→ no

Turn off TERMINAL-3, wait 5 seconds, then turn it back on again.

→ acknowledge

Using the cursor control key, move the cursor out from the left margin. Depress the return key, what appears on the screen?

→ nothing

What is displayed on the first line of TERMINAL-3?

→ terminal-ready

Is TERMINAL-3 connected securely to CABLE-3?

→ no

Connect TERMINAL-3 to CABLE-3.

→ acknowledge

Using the cursor control key, move the cursor out from the left margin. Depress the return key, what appears on the screen?

→ it-cr-and-colon

Problem(s):
TERMINAL-DISCONNECTED
TERMINAL-IN-LOCAL-MODE
resolved.
THANK-YOU

Acknowledgments
The authors would like to thank Michael Shorts, Jack Hymer, and Joe Gates for acting as domain experts during the development of this product. Their combined expertise forms the knowledge base that allows Schooner and its users to troubleshoot asynchronous datacom problems.

References