Tele-diagnosis: Remote monitoring of large-scale systems[1][2]

Somnath Deb, Sudipto Ghoshal, Venkata N. Malepati, and David L. Kleinman
Qualtech Systems, Inc.
6 Storrs Road, Suite 6
Willimantic, CT 06226
(860) 423-3659
deb@teamqsi.com

Abstract— Modern systems such as high rise buildings, nuclear power plants, manufacturing facilities, battlefields, etc. are all examples of highly connected network enabled systems. Many of these systems need to be monitored round the clock for high availability. Such systems typically consist of embedded sensors in networked subsystems that can transmit data to central (or remote) monitoring stations. In this paper, we present a distributed diagnosis solution that will not only monitor these sensor data streams, but also troubleshoot and diagnose subsystem failures, automatically, and in real-time. We present the software architecture that adapts TEAMS-RT – our real-time, embeddable diagnostic reasoner – into a distributed monitoring and diagnosis framework for tele-diagnosis of large, complex systems. This is followed by a description of the individual modules of the architecture, their interactions captured in time-sequence diagrams, and simulation results. Finally, we highlight some of our current and future research in the area.

TABLE OF CONTENTS

1. INTRODUCTION

Continuous *real-time failure detection and isolation* capability is essential to operation of high-availability safety critical systems. Prompt and accurate diagnosis of problems will:

- *improve operational safety*: The control room needs to be advised of the root causes, and their criticality, instead of being simply confronted with multiple alarms that seem to implicate many more subsystems. Such a capability will help support personnel respond to alarms quickly and more effectively.
- *improve availability by reducing time to diagnose failures*: The control personnel will not have to spend valuable time analyzing the alarms and trying to troubleshoot the system.
- *improve confidence in system serviceability*: The proposed solution will provide the system-wide self-test and monitoring capability, so that the health of the systems can be continuously and accurately assessed.

Thus, a real-time monitoring system will promote safety by providing a *continuous and accurate assessment of system health.* In most systems, the initial cost of the hardware and software of the monitoring system is easily offset by the savings in ground support and maintenance costs within a year [1]. Such savings in ground support costs are achieved by:

- *reducing manual diagnostic costs*: The monitoring system will automatically identify failures as and when they happen in normal operation. Thus, less time and money is spent in troubleshooting failures by support personnel. This also results in reduced turn-around times and improved system availability.
- *reducing "cannot duplicates"*: Certain failures manifest only under certain specific operational modes or stress conditions, e.g., at a certain resonant frequency. Such conditions cannot be duplicated off-line. Consequently, current maintenance procedures fail to diagnose them, resulting in re-certification of possibly faulty systems and recurring test expenses. The embedded monitoring system will be able to detect and isolate a failure when it manifests itself in actual operation.
- *facilitating condition-based maintenance*: The monitoring process can be combined with sophisticated signal-processing and trend analysis algorithms to identify *failing* components. This would result in maintenance-on-demand, and reduce unscheduled repairs during production runs.

[1] This research has been sponsored by the Army Research Office (DAAG55-98-C-0057) and NASA-ARC (NAS2-99049).
[2] 0-7803-5846-5/00/$10.00 © 2000 IEEE

To sum up, a real-time fault detection and isolation solution will not only make operations *faster, cheaper* and *better*, but also *safer*, reducing the likelihood of operational failures and disasters due to sudden failures, thereby improving system availability. Such a continuous health monitoring system must be able to collect sensor data, process it, and assess system readiness in real-time. In the event of failures, the fault should be isolated as quickly as possible, and the system should be immediately reconfigured to contain any damaging effect of the failure. If the failure is in a redundant subsystem, the reconfigured system can continue with the mission, possibly in a degraded mode. Otherwise, the mission objectives and available capabilities of the system should be evaluated, perhaps to modify the mission. Thus, the onboard diagnostics system should be responsive and decisive.

In an effort conducted with NASA-ARC in 1996, we had developed a model-based reasoning engine, TEAMS-RT [2,3,4], for the detection and isolation of multiple faults. The key features of TEAMS-RT are:

- Separation of the system-specific knowledge, captured in terms of models, from the fault-isolation methods. This allows for the same tool to be used on multiple systems using different models.

- Ability to diagnose multiple failures in fault-tolerant systems with multiple modes of operation.

- Ultra-compact memory requirements: only about 2 MB for two subsystems with 1000 failure sources and 1000 test points each.

- Excellent performance using low-end microprocessors (e.g., less than 200ms for the above-mentioned system on a 75MHz Pentium processor).

The TEAMS-RT reasoning engine is therefore an ideal choice for real-time embedded diagnosis. Indeed, we are utilizing it for health monitoring of diverse systems – from Helicopter engine [5] and transmission [6] to 1553 bus systems in the International Space Station [7]. However, embedding additional software in onboard systems and/or HUMS computers often introduce flight safety and validation concerns; the aerospace community is extremely reluctant to introduction of new hardware and/or software in flight approved systems. This has been a significant hindrance to the acceptance of embedded diagnosis solutions based on TEAMS-RT.

The International Space station, for example, is sensor rich. It, as well as most other NASA space systems, transmits voluminous amounts of sensor data to ground support systems (at NASA-Johnson Space Center, Houston, Texas) for health assessment. This data stream is near real-time, and consists of detailed sensor data from multiple subsystems on board the spacecraft. This presents an unique opportunity: if we can demonstrate a real-time *remote* monitoring solution that utilizes this telemetry data to monitor the health of the various subsystems, we can demonstrate the benefits of an onboard solutions, without having to actually install any software on the space station itself!

Moreover, sensor-rich systems, lacking a built-in health monitoring capability, are not rare. For example, OTIS has elevator systems, and Pitney-Bowes has copy machines, which are capable transmitting sensor data to remote service centers. Even the modern automobile has plenty of sensors, but only lack a wireless communication link. All of these systems could benefit from a *tele-diagnosis* capability, where remote data-streams from multiple subsystems are processed by offsite reasoners for diagnosis and prognosis of the remote system(s). Implementation of such a tele-diagnosis sever system is significantly cheaper than embedding reasoners in the subsystems, and would pay for itself by the savings realized from reduction of service calls by technicians.

In fact, in an increasingly connected world, it is not hard to imagine systems ranging from household appliances to sophisticated aircraft systems routinely connecting to remote reasoning services for periodic health assessment, prognosis and diagnosis. In such a system, or in our current project involving remote monitoring of the space station, the common issues to be addressed in designing a Tele-diagnosis architecture include:

- How will remote sensors find monitoring services or reasoners? Consider a dynamic network, *m* subsystems to be monitored by a bank of *n* computers running multiple TEAMS-RTs. There should be some sort of matchmaking service for sensors and monitors to find each other, while balancing the computational load across monitoring resources.

- How do you make sure sensor results are not dropped because TEAMS-RT is busy processing the previous problem? Real-time operations require the sensors be able to deliver the data to TEAMS-RTs without having to wait or block. There should be some low-latency buffers in the network to store and forward the data to TEAMS-RT.

- How do you manage these sessions, i.e., a sensor connecting to a TEAMS-RT, disconnecting, and then connecting back again? Some session management capabilities are needed.

- How do we ensure distribution of information? For example, the subsystem TEAMS-RTs needs to monitor only the relevant subsystem, but the supervisor TEAMS-RT should be able to monitor the

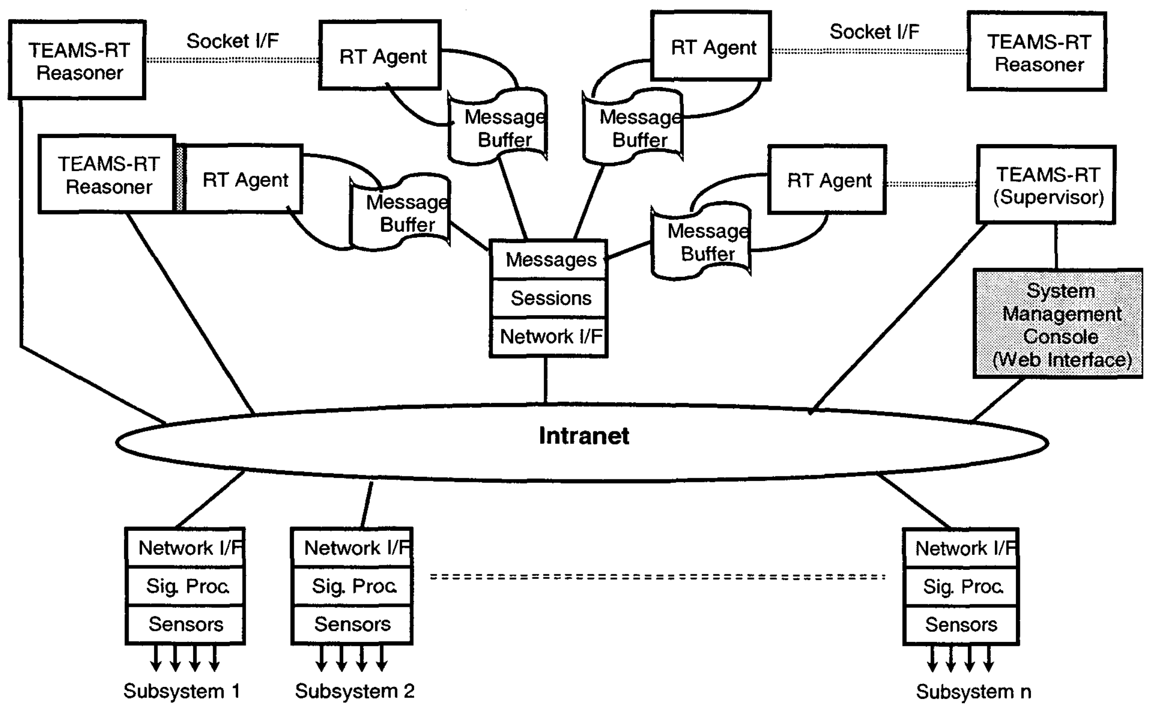


Figure 1: The network-based, distributed diagnosis architecture leveraging the Intranet

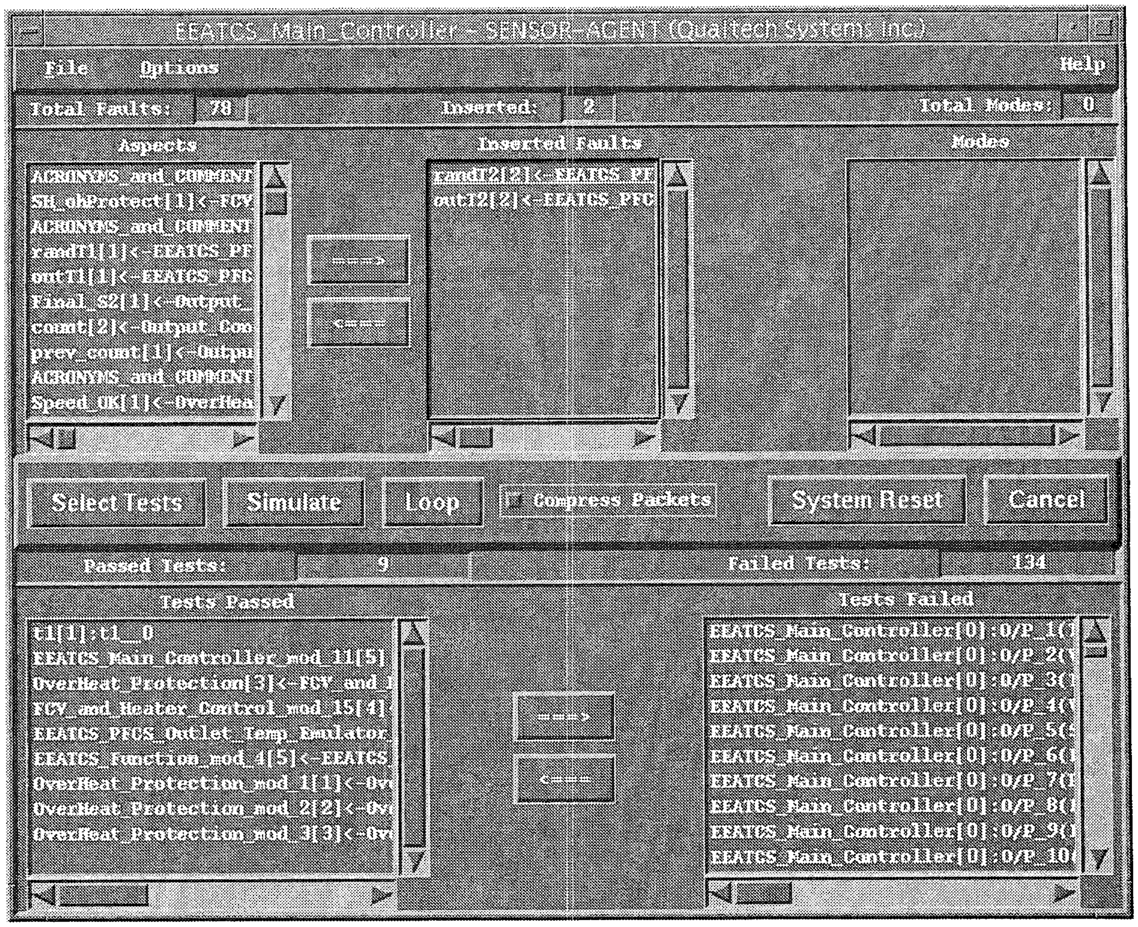
entire system and compute a system-level diagnosis. This calls for a middle layer, with some policies on access privileges.

A scalable tele-diagnosis solution, using a network of TEAMS-RT reasoners, is also essential to monitoring large safety critical systems (e.g., nuclear power plants, wind tunnel systems, nuclear submarines etc.) that are too complex to be monitored by a single TEAMS-RT reasoner. Such a solution would allow for the efficient separation of the computational resources associated with the operation of the large-scale system from the monitoring and diagnosis resources.

2. Architecture Overview

In order to ensure a truly distributed, multi-computer tele-diagnosis system, we came up with an architecture as outlined in Fig.1. This distributed processing architecture is inspired by ToolTalk [9] and CORBA (Common Object Request Broker Architecture) [10,11]. The issues regarding scalability and inter-process communication as identified in the previous section are typically addressed by CORBA. However, CORBA is too complex to embed in small computer systems, such as onboard helicopters or automobiles. It is intended for problems that are much more generic and complex than our requirements. In addition, deployment of a CORBA-based system has a very high initial cost. The ToolTalk architecture is a simpler middle-layer architecture that seemed appropriate for our needs. The ToolTalk service enables independent applications to communicate with each other without having direct knowledge of each other. Applications create and send ToolTalk messages to communicate with each other. The ToolTalk service receives these messages, determines the recipients, and then delivers the messages to the appropriate application. The ToolTalk architecture can be summarized as follows:

1. It is a store-and-forward messaging architecture.
2. ToolTalk defines two message types: Notice (advisory) and Request (Call for action)
3. Likewise, it defines two application types – Observer and Handler.
4. There can be one and only one handler but many observers for any given message.
5. Applications register message types they can handle and or observe.
6. The broker (or middle layer) buffers all messages and notifies the appropriate handler and observers, starting the relevant application process if necessary.



```
Total Faults: 78          Inserted: 2          Total Modes: 0

Aspects                          Inserted Faults            Modes
ACRONYMS_and_COMMENT             randT2[2]<-EEATCS_PF
SH_ohProtect[1]<-FCV             outT2[2]<-EEATCS_PFC
ACRONYMS_and_COMMENT
randT1[1]<-EEATCS_PF
outT1[1]<-EEATCS_PFC
Final_S2[1]<-Output_
count[2]<-Output_Con
prev_count[1]<-Outpu
ACRONYMS_and_COMMENT
Speed_OK[1]<-OverHea

Passed Tests: 9                                 Failed Tests: 134

Tests Passed                                    Tests Failed
t1[1]:t1__0                                     EEATCS_Main_Controller[0]:O/P_1(
EEATCS_Main_Controller_mod_11[5]                EEATCS_Main_Controller[0]:O/P_2(
OverHeat_Protection[3]<-FCV_and_                EEATCS_Main_Controller[0]:O/P_3(
FCV_and_Heater_Control_mod_15[4]                EEATCS_Main_Controller[0]:O/P_4(
EEATCS_PFCS_Outlet_Temp_Emulator                EEATCS_Main_Controller[0]:O/P_5(
EEATCS_Function_mod_4[5]<-EEATCS                EEATCS_Main_Controller[0]:O/P_6(
OverHeat_Protection_mod_1[1]<-Ov                EEATCS_Main_Controller[0]:O/P_7(
OverHeat_Protection_mod_2[2]<-Ov                EEATCS_Main_Controller[0]:O/P_8(
OverHeat_Protection_mod_3[3]<-Ov                EEATCS_Main_Controller[0]:O/P_9(
                                                EEATCS_Main_Controller[0]:O/P_10
```

Figure 2: User interface of the demonstration version of Sensor Agent

7. The ToolTalk service manages sessions and pairs up Request messages with appropriate Handlers, starting or restarting a new session if necessary.

Our Broker module leverages the distributed message handling capabilities of the ToolTalk architecture, with several modifications and enhancements. Conceptually, we added another category of application type, namely the Generator in addition to the existing categories of Handlers and Observers. The Generator module is the one that initiates or generates the message that requires to be passed to a Handler. The category was required since not all messages generated in our system will be through user interaction. The roles of these three application types are interchangeable and depending on the message type, the Broker decides the Handler of the message generated, notifies the Handler and any Observers, if they are registered for the message. Thus, the broker performs the job of the Object Request Broker (ORB) in the CORBA context. All messages and their content are completely transparent to the broker, unless the message is one for which the broker has registered with itself as a Handler or Observer.

3. COMPONENTS OF THE ARCHITECTURE

The tele-diagnosis architecture presented in Fig. 1 consists of 5 major components and complex interactions between them. In the following, we identify the functionality of each component, and describe their interactions in simplified sequence diagrams (see Figs. 4 and 5)

Sensor Agent

The Sensor-Agent is responsible for acquisition of sensor signals, signal conditioning (if necessary) and pass/fail decision on the resulting data. The resulting test results are periodically sent to the broker for subsequent processing by the reasoner. In addition, the Sensor-Agent may retrieve commands to be executed from supervisor TEAMS-RT and pass it on to the underlying control logic. Such commands include mode changes and drill-down tests that will allow quicker or more precise diagnosis.

During normal operation a sensor client performs the following steps:

1. When initiated, it connects and registers with the broker. The registration process involves identifying the subsystem (and hence, the corresponding model) that the sensor client is attached to.

2. The broker returns a session ID after successfully connecting to a RT-Agent and confirming that a TEAMS-RT reasoner has successfully initialized with the right model.

3. The sensor client loads the list of sensors it is supposed to monitor, identify system modes and sensor rates/timing information etc., and start retrieving sensor data. It may invoke the preprocessing module's signal processing routines and test decision rules to convert test data into test results.

4. Optionally, the sensor client can also send raw data for further processing by other applications invoked by the broker.

5. It also retrieves test commands such as request for drill down tests from the broker.

6. It also participates with the broker in session control – i.e., honoring or initiating START, STOP, SUSPEND, RESUME etc. modes as necessary.

We have currently implemented a sensor client GUI for demonstration purpose (Fig. 2). The user starts up the GUI, selects a model and seeds one or more faults. The GUI then sends periodic test results to the broker. In actual implementation, the sensor client will be an embedded application that will poll sensors present in the subsystem and send test results to the broker.

Broker

The broker is responsible for the following four functions:

1. Store and forward messages to proper applications. This includes passing test results from sensor clients to TEAMS-RT and supervisor TEAMS-RT via their respective agents as well as commands from Supervisor TEAMS-RT and the user to various RT-Agents and sensor clients.

2. It is responsible for identifying (and starting, if necessary) the appropriate Handler and Observer applications, e.g., RT-Agents.

3. It is responsible for managing sessions and the associated shared memory buffers. It also distributes the computation load (basic load balancing) across the network by invoking RT-Agents on remote computers.

4. It enforces access privileges to the shared data. For example, the supervisor can plant commands for any sensor client or shared RT-Agent, whereas individual RT-Agents can only plant commands for its sensor client.

A broker will therefore perform the following steps:

1. Accept new connections and assign session IDs. Connections can be originated by,
    - Sensor-Agents – delivering test results.
    - RT-Agents passing messages to neighboring RT-Agents to co-ordinate their diagnosis.
    - Supervisor TEAMS-RT monitoring and supervising the distributed diagnosis

2. Create/attach/refer shared memory based on Session ID and client type/history and copy messages to the appropriate shared memory buffers

3. Register and/or launch the Handler application and also notify the Observer applications, if any.

4. Put any incoming messages from Generator applications, like Sensor-Agents or Supervisor TEAMS-RTs in data structure in shared memory.

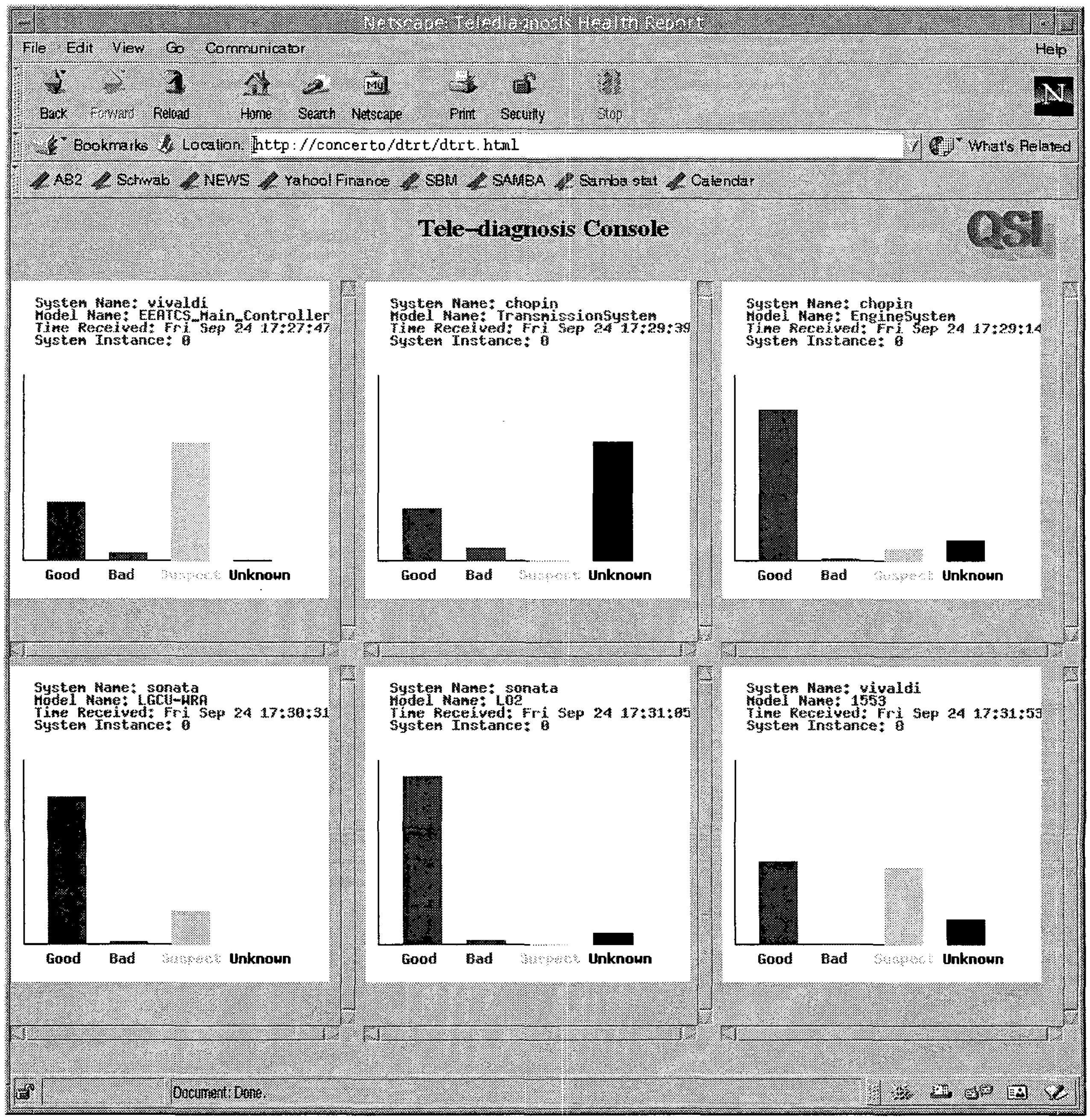
5. Check for messages/commands to be delivered to client

6. Check and maintain data integrity in shared memory. This includes use of semaphores for locking to prevent simultaneous write access by multiple applications

7. Distribute messages by copying incoming messages to multiple buffers as appropriate for client access privileges.

8. Perform garbage collection – i.e., clear buffers that has been read, free shared memory segments etc.

In our present implementation, the broker can only accept connections from sensor clients and invoke the RT-Agent. Currently, all communication between broker and TEAMS-RT reasoner is via the RT-Agent. We are presently upgrading the broker and message formats for connections from the TEAMS-RT reasoners and supervisor TEAMS-RT. Fig. 3 presents a screen-dump of the web-accessible diagnostic console with health report from 6 different sensor clients/models from three different computers. The user can view the system health summary as presented in histograms of Good, Bad, Unknown and Suspected components, or click on each of the bars to view the list of components comprising the diagnosis.


Netscape: Telediagnosis Health Report
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://concerto/dtrt/dtrt.html What's Related
AB2 Schwab NEWS Yahoo! Finance SBM SAMBA Samba stat Calendar
Tele-diagnosis Console
QSI
Good Bad Suspect Unknown
Document: Done.

System Name: vivaldi
Model Name: EEATCS_Main_Controller
Time Received: Fri Sep 24 17:27:47
System Instance: 0

System Name: chopin
Model Name: TransmissionSystem
Time Received: Fri Sep 24 17:29:39
System Instance: 0

System Name: chopin
Model Name: EngineSystem
Time Received: Fri Sep 24 17:29:14
System Instance: 0

System Name: sonata
Model Name: LGCU-WRA
Time Received: Fri Sep 24 17:30:31
System Instance: 0

System Name: sonata
Model Name: L02
Time Received: Fri Sep 24 17:31:05
System Instance: 0

System Name: vivaldi
Model Name: 1553
Time Received: Fri Sep 24 17:31:53
System Instance: 0

Figure 3: The web console for distributed system health management.

RT-Agent

The RT-Agent is responsible for the following two functions:

1. To serve as a abstraction layer between the broker and the TEAMS-RT reasoner so that it can be adapted to this architecture with minor modifications. It periodically checks with the broker for new test results and commands to process, and calls TEAMS-RT with the appropriate function interface.

2. To invoke TEAMS-RT on remote computers and pass messages over socket, if necessary. This would allow the distribution of TEAMS-RTs across the network.

A RT-Agent will therefore perform the following steps:

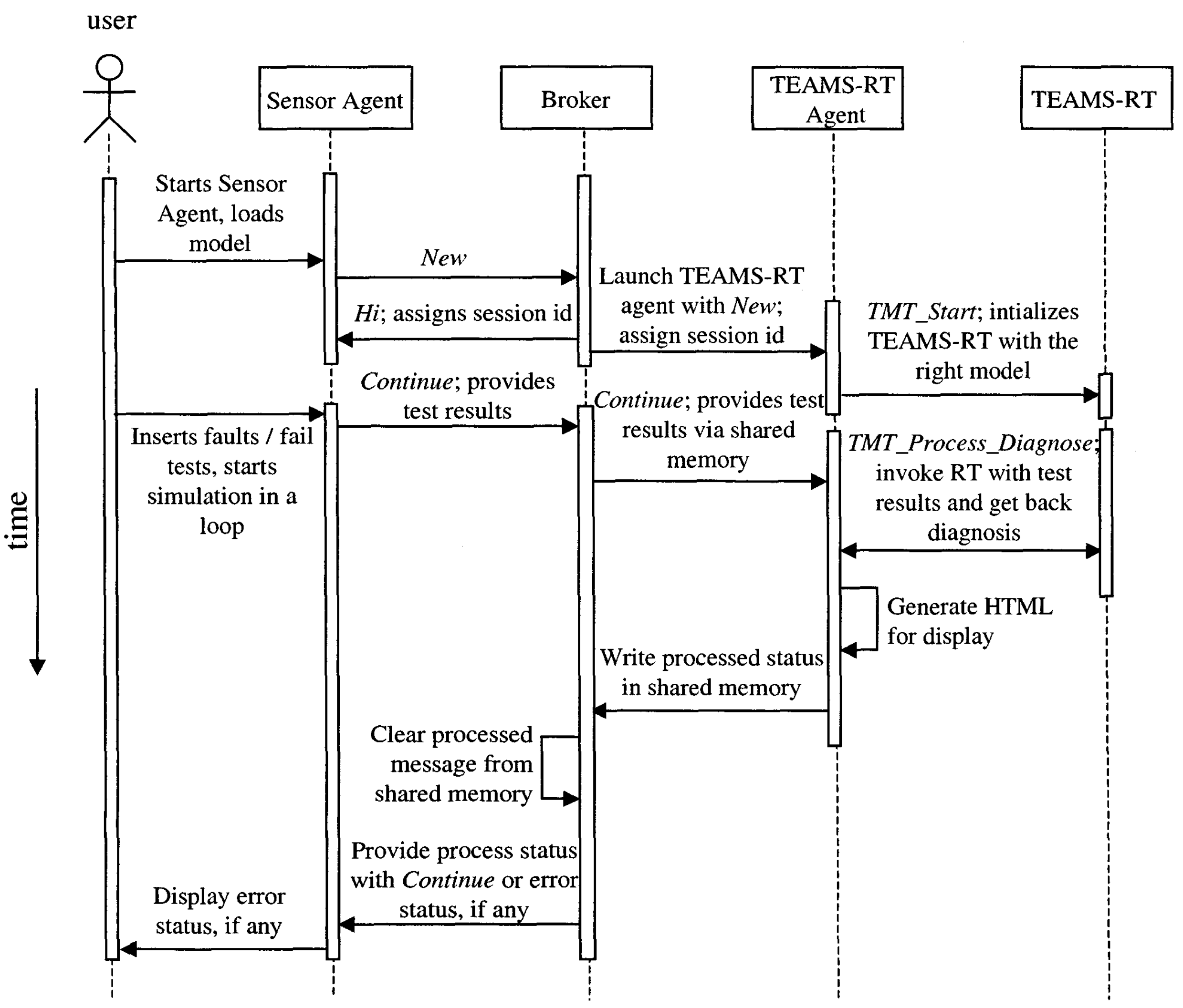

user
Sensor Agent
Broker
TEAMS-RT Agent
TEAMS-RT
Starts Sensor Agent, loads model
New
Hi; assigns session id
Launch TEAMS-RT agent with New; assign session id
TMT_Start; intializes TEAMS-RT with the right model
Continue; provides test results
Continue; provides test results via shared memory
Inserts faults / fail tests, starts simulation in a loop
TMT_Process_Diagnose; invoke RT with test results and get back diagnosis
time
Generate HTML for display
Write processed status in shared memory
Clear processed message from shared memory
Provide process status with Continue or error status, if any
Display error status, if any

Figure 4: UML Sequence Diagram of some of the message flow between the different modules as currently implemented

1. When the broker starts a RT-Agent, it invokes a TEAMS-RT reasoner (either locally, or in a remote computer) and installs itself as a handler for the sensor-client.

2. Every t seconds (where t is a predefined sampling interval based on sensor data rates, typically t = 2 s) , it checks the shared memory segment for test and commands to process.

3. It processes the message and then invokes the TEAMS-RT reasoner with appropriate data.

4. The agent may communicate directly with TEAMS-RT (i.e., make a function call) or over a socket interface. In the latter case, the TEAMS-RTs can be running on different computers.

When TEAMS-RT completes the action (successfully or with an error condition), the RT-Agent writes back the status of the message processing by TEAMS-RT in the appropriate location of the shared memory segment for the broker. In the event of an error condition, the broker relays the error message to the sensor client for display to the user.

It also employs locking and access-protection (using semaphores etc.) when accessing the shared memory, and coordinates with the broker to free up memory once the message has been read. Currently the RT-Agent is also responsible for obtaining the current diagnostic status from its associated TEAMS-RT and generating the

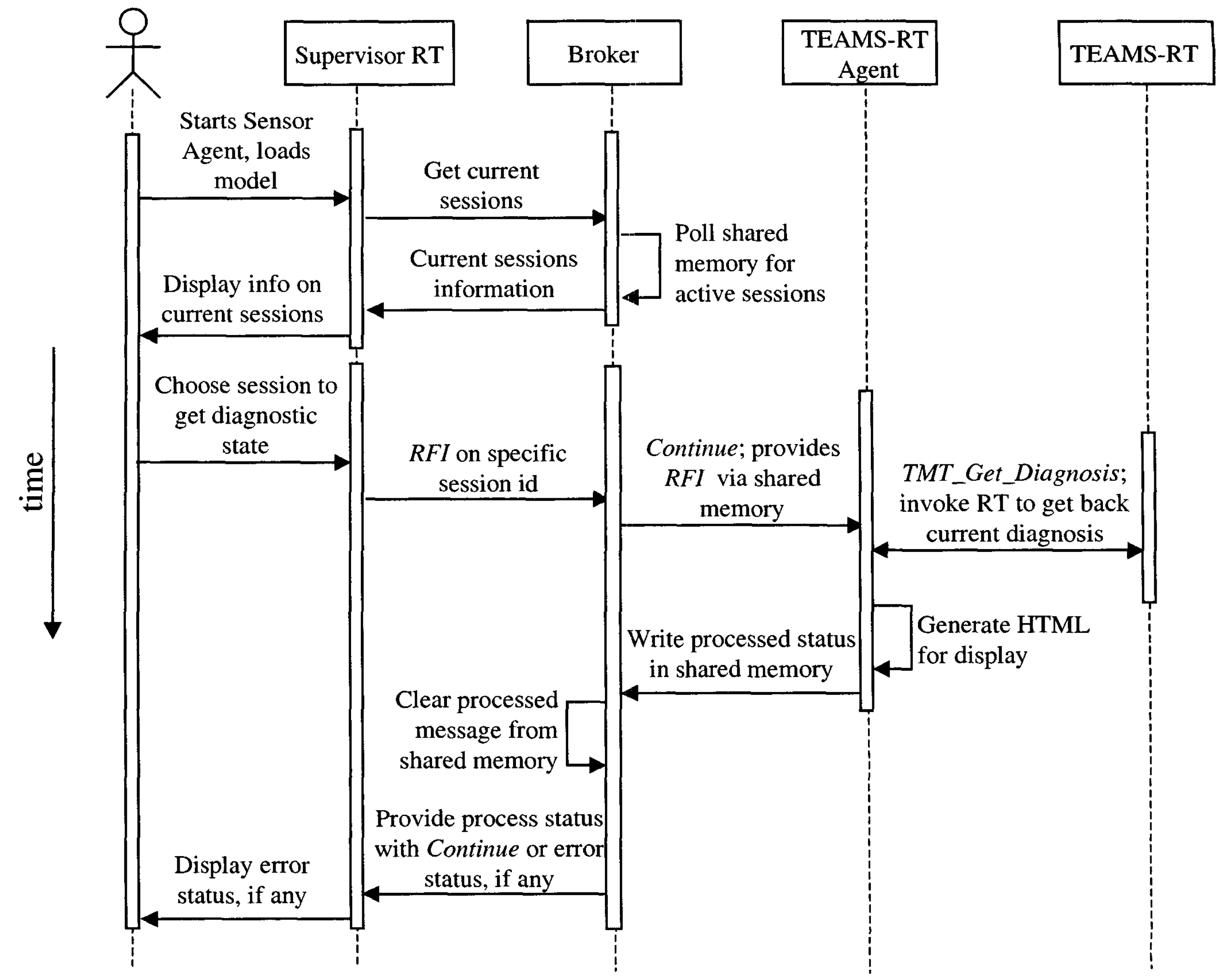

Supervisor RT
Broker
TEAMS-RT Agent
TEAMS-RT
time
Starts Sensor Agent, loads model
Get current sessions
Poll shared memory for active sessions
Current sessions information
Display info on current sessions
Choose session to get diagnostic state
RFI on specific session id
Continue; provides RFI via shared memory
TMT_Get_Diagnosis; invoke RT to get back current diagnosis
Generate HTML for display
Write processed status in shared memory
Clear processed message from shared memory
Provide process status with Continue or error status, if any
Display error status, if any

Figure 5: UML Sequence Diagram of some of the message flow between the Supervisor TEAMS-RT and other modules to be implemented.

hypertext documents (HTML) for constant display on the web (Fig. 3). Eventually, we want to develop a dedicated System Management Console application as part of the Supervisor TEAMS-RT which will have the user interface for the user to obtain the diagnostic status of all or any subsystem and the global diagnosis from a single console.

TEAMS-RT Reasoner

The TEAMS-RT reasoner is based on our current TEAMS-RT product [4,8], and will be modified in the following ways:

1. We will add a socket interface to TEAMS-RT so that it can interface with the RT-Agent over the network. This will also provide the ability for TEAMS-RT reasoner to connect to the broker and deliver messages for neighboring TEAMS-RTs to coordinate distributed diagnosis.

2. We will enhance the TEAMS-RT algorithms for distributed diagnosis as is outlined in [8].

Supervisor TEAMS-RT

The supervisor TEAMS-RT is a new derivative module of TEAMS-RT and TEAMATE, designed to address the need for a global reasoner with a system wide view. The supervisor TEAMS-RT can simply be a tool that collects and fuses the diagnosis of the individual subsystem TEAMS-RTs, or it could have the capability to shadow monitor a subsystem TEAMS-RT, perhaps employing deeper reasoning algorithms that are non/pseudo real-time.

Thus, the supervisor TEAMS-RT has the following additional capabilities:

1. It has access privileges to read any and all test results of subsystem TEAMS-RT. The supervisor TEAMS-RT is therefore installed as an Observer process for all messages. This allows it access to all the necessary information to perform shadow reasoning on any subsystem.

2. It has access privileges to connect to the broker and plant commands for any subsystem TEAMS-RT and sensors. Such commands include:
   - Instruction to report intermediate diagnosis to supervisor
   - Instruction to change component states in diagnosis, e.g., change the state of an faulty component to good following a repair
   - Instructions for session management – e.g., SUSPEND, RESUME, etc.
   - The supervisor TEAMS-RT will also have a Web-enabled interface to implement the system management console. Users will then be able to query health status of subsystems and control/manage the distributed diagnosis process.

Message Formats

Our implementation of the distributed diagnosis architecture is based on message passing. Each message will have the following parts to make parsing and distribution of messages easier.

1. The header section consists of the following:
   a. A source ID consisting of the sub-system name, model name, and the hostname. In the context of message passing in a 1553 bus system, all three entries could simply correspond to the subsystem ID.
   b. The client type: one of SENSOR, TEAMS-RT, BROKER, or SUPERVISOR
   c. A Destination ID similar to the source ID above. However, this field is optional, and required mainly for inter-TEAMS-RT communications
   d. A Session ID, which is assigned by the broker
   e. When a new session is initiated, e.g., by a sensor client, it is set to -1 implying the broker needs to assign a new session number. A session ID of 0 signifies an unknown session ID (e.g., if a sensor client was restarted and wanted to resume a session) and the broker tries to recover the session ID based on model name, system name etc.
2. The session command section, if present, will consist of instructions such as START, STOP, SUSPEND, RESUME, RESET and EXIT
3. The diagnosis command section will consist of instructions such as PROCESS_DATA, PERFORM_TEST, CHANGE_MODE, REPORT _DIAGNOSIS etc. Each diagnosis command is paired with a message block, and there can be multiple such pairs in a given message.
4. The message block contains data associated with the diagnosis command, such as Test name and number, Test outcome or sensor Data block, Fault name and number, Fault status, etc.

Sequence Diagrams

Figure 4 shows some of the message flows among the different components in the form of a Unified Modeling Language (UML) sequence. The sequence shows the message flow, as currently implemented, when the user starts up the Sensor-Agent, loads a model, inserts faults or fails tests and runs the simulation. Figure 5 shows a UML sequence diagram of the user utilizing the Supervisor TEAMS-RT interface to retrieve the current active sessions and obtaining the diagnostic state of a particular session.

4. SIMULATION RESULTS AND CONCLUSIONS

We ran our tele-diagnosis server, consisting of the Broker, RT-Agent, and TEAMS-RT, on a Sun SS20/502 system with 50 Mhz SuperSPARC processor, 224 Mb of RAM. We ran the Sensor-Agent clients on both Sun boxes and Windows NT PCs with various hardware configurations. We also ran a web server to serve the tele-diagnosis console (see Fig. 3) on the SS20. We collected several performance-related metrics for the different modules and the CPU processing times and memory requirements for the messages on the server-side. We also performed simulations with dozens of concurrent clients to verify the scalability of our architecture.

Table 1 shows some of the preliminary results obtained for the Broker, RT-Agent and the TEAMS-RT modules for different diagnostic models. The objective here is to determine the computational load of the components of the tele-diagnosis server for problems of various sizes. Please note that all processing times were for a 5 year old 50MHz processor, which is about 7 times slower than a Ultra 5 workstation costing under $3000. We picked a slow computer for the tele-diagnosis server because the time routines were unreliable in the sub-ms range, and the run-time of the broker could not be reliably measured in the faster computers. The results clearly demonstrates even an old SS20/502 could provide tele-diagnosis to ten's of clients, provided the reasoners are run on remote (faster) computers for models consisting of more than 1000 failure sources (using socket connections between the RT-Agent and TEAMS-RT, as outlined in Fig. 1).

Table 1: Simulation results for 7 different models of varied complexity

Model	Number of Tests Pass / Fail	Number of Faults inserted / total modeled	RT-agent CPU run time in ms	Broker CPU run time in ms	TEAMS-RT CPU run time in ms
1553	59 / 2	2 / 174	250	10	< 5
Transmission system	46 / 5	2 / 160	210	9	< 5
EEATCS	9 / 134	2 / 74	250	12	< 5
Documatch	175 / 5	2 / 259	230	7	< 5
LO2	329 / 39	3 / 167	300	7	< 5
Engine System	274 / 32	3 / 255	300	7	< 10
LGCU-WRA	1003 / 316	4 / 2080	500	12	< 250

During each simulation, the diagnostic model was loaded onto the Sensor-Agent, several faults, ranging from 2 for the smallest model, to 4 for the largest, were inserted. The number of tests and their outcomes resulting from these faults was noted and a continuous simulation was started. The Sensor-Agent would then send the test results every 2 seconds to the Broker. The time required for processing of this data stream by the Broker, RT-Agent and TEAMS-RT reasoner were averaged for each data set. All averages are based on at least 50 runs. The Broker memory footprint was about 12MB for all the runs, the RT-Agent memory requirements varied between 12 and 17MB.

From these preliminary results in Table 1, it is evident that the Broker's memory and CPU requirements are nearly independent of the size of diagnostic model. This is probably due to the fact that the Broker has fixed size buffers and copying incoming data to these buffers are relatively simple operations. In any case, the low response time and insensitivity to model size are important desirable features from scalability considerations. Since the Broker requires very low processing time (< 12 ms) and CPU utilization (< 0.5%) for each client, it appears that the Broker can easily scale up to 50 client connections, even for the largest models. The scalability of the Broker module is critical for an optimal real-time performance of the tele-diagnosis application. Further tests with multiple clients connecting are being conducted for accurate quantification of the scalability of the Broker module.

Scalability for handling multiple clients, however, is not an issue with RT-Agent or TEAMS-RT as they service only one subsystem, i.e., run only one diagnostic model at any given time. This results from the fact that the Broker handles all session management for incoming connections, and passes only the client (sub)system data to the appropriate reasoner. However, the computational complexity increases super-linearly with the model size (~$O(n^2)$). Therefore, for concurrent diagnosis of multiple (i.e., 5 or more) large models like the LGCU-WRA with over 2000 failure sources and over 1300 tests, faster and/or multiprocessor computers (than the 50Mhz SS20) are necessary.

Currently, TEAMS-RT is implemented as a shared library for RT-Agent. The message flow through the agent requires the agent to process it and generate the correct input for the TEAMS-RT diagnostic kernel. In addition, the agent also generates the output HTML documents with embedded GIF histograms (see Fig. 3) to report system health. Since both these tasks are dependent on the size of the diagnostic model, the processing time of the agent also increases with the model size. In subsequent implementations, where Supervisor TEAMS-RT will be the user interface to the Tele-diagnosis system, the generation of the output documents will no longer be a responsibility of the RT-Agent. This should eliminate a large chunk of processing time associated with disk I/O and GIF/HTML generation. In addition, for enhanced load-distribution, the RT-Agent and TEAMS-RT will also have remote communication protocols such as sockets built in. These design changes will substantially improve the real-time performance of both the RT-Agent and TEAMS-RT, and improve scalability of our Tele-diagnosis solution.

Another issue that needs to be addressed is the effective utilization of network bandwidth for the Sensor-Agent data packets. To reduce the bandwidth requirement we are incorporating a packet compression routine for the tele-diagnosis system that uses the standard Ziv-Lempel data compression algorithm. The data compression is especially useful for large systems, resulting in a 70% reduction in packet size. In addition, we are also in the process of developing algorithms for event compression, wherein Sensor-Agents will transmit only changes in test outcomes, instead of repeating tests with identical results.

Since the test results are likely to change only when the system changes state (i..e, enters a new mode of operation or develop failures), this is likely to substantially reduce the bandwidth requirement.

In conclusion, we believe we are on track to building a credible scalable tele-diagnosis server for simultaneous remote diagnosis of multiple (sub)systems of varied complexity and origin. This technology helps us leverage our integrated toolset [12] consisting of TEAMS modeling environment, TEAMS-KB information management system and TEAMS-RT and TEAMATE reasoner modules to solve diagnostic needs of connected systems of the new millennia.

5. REFERENCES

[1] *Sikorsky Aircraft Corporation Internal Cost Benefit study for the COSSI Project, 1998*

[2] *Patterson-Hine, A., Kulkarni, D., Deb, S. and Wang, Y. "Automated System Checkout to Support Predictive Maintenance for the Reusable Launch Vehicle,"* Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, *San Diego, October 11-14, 1998.*

[3] *Kulkarni, D., Patterson-Hine, A., Holthaus, M., Deb, S., and Pattipati, K.R., "Degradation Detection and Testability Analysis in Propulsion Checkout and Control System,"* Proceedings of the Aerotech'95, *Los Angeles, CA, Sept. 1995.*

[4] *Mathur, A., Deb, S., and Pattipati, K. R., "Modeling and Real-Time Diagnostics in TEAMS-RT,"* Proceedings of the American Control Conference, *Philadelphia, June 24-26, 1998.*

[5] *"Joint Advanced Health and Usage Monitoring System - Advanced Concept Technology Demonstration", Phase I, Final Report, Sikorsky document no. SER 521365, August, 1998. Also, http://www.dt.navy.mil/jahums/ JAHUMS Project Homepage.*

[6] *An Onboard Real-time Aircraft Diagnosis and Prognosis System. Technical Progress Report on NAS2-99048, October 26, 1999.*

[7] *A Systematic Integrated Diagnostic Approach to Software Testing. Technical Progress Report on NAS2-99049, September 27, 1999.*

[8] *Deb, S., Mathur, A., Willett, P.K., and Pattipati, K.R. "De-centralized Real-time Monitoring and Diagnosis,"* Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, *San Diego, October 11-14, 1998.*

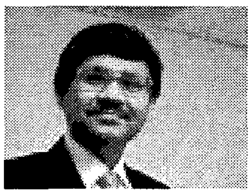
[9] *Common Desktop Environment 1.0 Tooltalk Messaging Overview (Common Desktop Environment 1.0), CDE Documentation Group, Addison-Wesley Pub Company, N.Y., 1995.*

[10] http://www.omg.org/corba

[11] *CORBA Fundamentals and Programming, Ed. Jon Siegel, John Wiley & Sons, N.Y., 1996.*

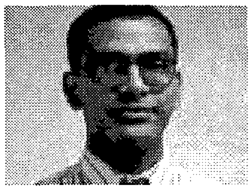
[12] *Deb, S., Pattipati, K.R. and Shrestha, R., "QSI's Integrated Toolset", in Proc. IEEE Autotestcon, Anaheim, CA, pp. 408-421, Sept. 22, 1997.*

6. BIOGRAPHIES



Somnath Deb, *Chief Scientist at Qualtech Systems, Inc., received the B.Tech. degree in Electrical and Electrical Communications Engineering from IIT-KGP, India (1987) and M.S. and Ph.D. degrees in Control and Communication Systems from the University of Connecticut in 1990 and 1994, respectively. Dr. Deb's research has included the development of advanced optimization algorithms for systems testability analysis and improvement and multisensor multitarget tracking and data association. He has published over 30 journal and conference papers. He received the Best Technical Paper Awards at the 1990 and 1994 AUTOTEST Conferences for his work on tools for system testability analysis and multi-signal modeling. He is a senior member of IEEE.*

Sudipto Ghoshal, *is the Principal Research Engineer at Qualtech Systems, Inc. He received his B.Tech degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur, India in 1989, the M.S. and Ph.D. degrees in Biomedical Engineering from the University of Connecticut, Storrs in 1991 and 1997, respectively. Prior to joining Qualtech Systems, Dr. Ghoshal was a Senior Software Engineer at Netscape Communications Corporation, Mountain View. Dr. Ghoshal's research interests include understanding the signal processing of neural systems and development of tools and strategies for efficient system test and diagnosis. At Qualtech Systems, he is primarily involved in developing a web based system diagnostic and training tool, TEAMATE, using Java and CORBA and is involved in standard related efforts in the areas of diagnostics. He was also the lead developer in the development of a web based Reusable Test Library software using Java and JDBC (Java Database Connectivity).*



Venkata N. Malepati, *Senior Software Engineer at Qaultech Systems Inc., received his Bachelor's Degree in Electronics & Communication Engineering from Gulbarga University, India in 1993 and M.S degree in Electrical and Systems Engineering from the University of Connecticut, Storrs in 1999. He is primarily involved in developing a tool for software testing and validation, and development of QSI's real-time system diagnostic engine TEAMS-RT.*

David L. Kleinman, *is the Director of Research and Development at Qualtech Systems, Inc. Dr. Kleinman received the B.E.E. degree in Electrical Engineering from Cooper Union in 1962, the M.S.E.E. degree in Electrical Engineering from Massachusetts Institute of Technology in 1963, and the Ph.D. in Control Systems, Massachusetts Institute of Technology, 1967.*

Dr. Kleinman is internationally known for his work on manual control, human decision-making research, and computational algorithm development. He is a pioneer in the application of modern control and estimation theory to develop and validate an analytical model for describing human control and information processing performance in manned vehicle systems. In 1994, he was elected a Fellow of the IEEE. In various industrial positions and academic positions, he has led applied research projects in both manual control and automatic control. Examples include fixed-wing aircraft, helicopters and submarine control. By way of these applications and other applications, he has established credibility in applied digital control, parameter estimation, Kalman filtering, and simulation. His efforts in human decision-making have involved modeling operator performance and response characteristics in real-time multi-task sequencing and scheduling problems. His current research, which is funded through grants from the Office of Naval Research and the National Science Foundation, is in the area of multi-human decision-making in dynamic environments, and coordination in human teams. Dr. Kleinman has done extensive research in the area of team decision making within military Command and Control contexts.