REMOTE DIAGNOSIS SERVER ARCHITECTURE

Dr. Somnath Deb and Dr. Sudipto Ghoshal
Qualtech Systems, Inc., Suite 501, 100 Great Meadow Road, Wethersfield, CT 06109
Phone: (860) 257-8014, Email: deb@teamqsi.com

ABSTRACT

Modern systems such as the Space Shuttle, the International Space Station, or nuclear power plants are examples of mission critical systems that need to be monitored around the clock. Such systems typically consist of embedded sensors in networked subsystems that can transmit data to central (or remote) monitoring stations. At Qualtech Systems, we are developing a Remote Diagnosis Server (RDS™) to implement a remote health monitoring system based on received data from such systems. RDS™ can also be used to provide online monitoring of sensor-rich, network capable systems such as jet engines, building heating-ventilation-air-conditioning systems, and automobiles.

This paper presents the various components and architecture of the RDS™. The RDS™ is built on a three-tier architecture with a "Broker" application in the middle layer, and multiple TEAMS-RT™ and TEAMATE™ based reasoners at the backend. The client layer consists of sensor agents that collect test results and transmit them over a message-passing network, or technicians with web browsers being guided through intelligent troubleshooting sessions. A database in the backend, TEAMS-KB™, is used to manage models and content, and collect diagnosis logs for data mining. The solution scales easily to hundreds of sessions in any modern workstation or server.

Keywords: RDS™, Remote Diagnosis Server, scalability, online monitoring, guided troubleshooting, multisignal modeling, thin clients, telediagnosis, remote diagnosis.

1 INTRODUCTION

As technology advances, there is a significant increase in the complexity and sophistication of systems. Consequently, system monitoring and troubleshooting presents formidable challenges to manufacturers and end-users of sophisticated systems, such as combat aircraft, International Space Station, and reusable launch vehicles (RLVs). Such systems, consisting of complex interplay of electronic, electromechanical and hydraulic subsystems, can no longer be efficiently maintained based on old and often inaccurate, technical manuals and chivalry and heroism of field maintenance personnel. Increasingly, the complexity of the problem is overwhelming the reasoning capacities of even the most seasoned technician, while nineteen-year-olds are asked to maintain fleets of complex, state of the art equipment.

Motivated by the need for an integrated process for system maintenance and diagnostics in complex systems, current research at QSI is focused on a model-based diagnosis approach, where knowledge about system is captured in multisignal model [1]. Ideally, such models will be developed early at the design stage to perform diagnosis analysis, and evolve with the system.

We have developed a comprehensive tool set that uses this model of the system (see Figure 1) to perform Design for Testability (DFT), Failure Modes and Effects Analysis (FMEA), reliability analysis, online monitoring and diagnosis, flight line and depot level maintenance, and maintenance management, data logging, mining, configuration control etc. Our integrated diagnosis tool set [2] consists of:

- TEAMS™: Testability assessment and improvement (DFT), reliability analysis, Failure Modes, Effects and Criticality Analysis (FMECA) and pre-computed diagnostic test strategy generation in a variety of forms (e.g., SGML-based Interactive Electronic Technical Manual);
- TEAMS-RT™: on-board diagnostics, health and usage monitoring systems;
- TEAMATE™: Portable Intelligent Maintenance Aids (PIMAs) with interactive electronic technical manuals and multi-media animation, dynamic TPSs for ATEs.
- TEAMS-KB™: Scheduled and unscheduled maintenance and diagnostics data collection, statistical data analysis and data mining for trend and anomaly detection/isolation.

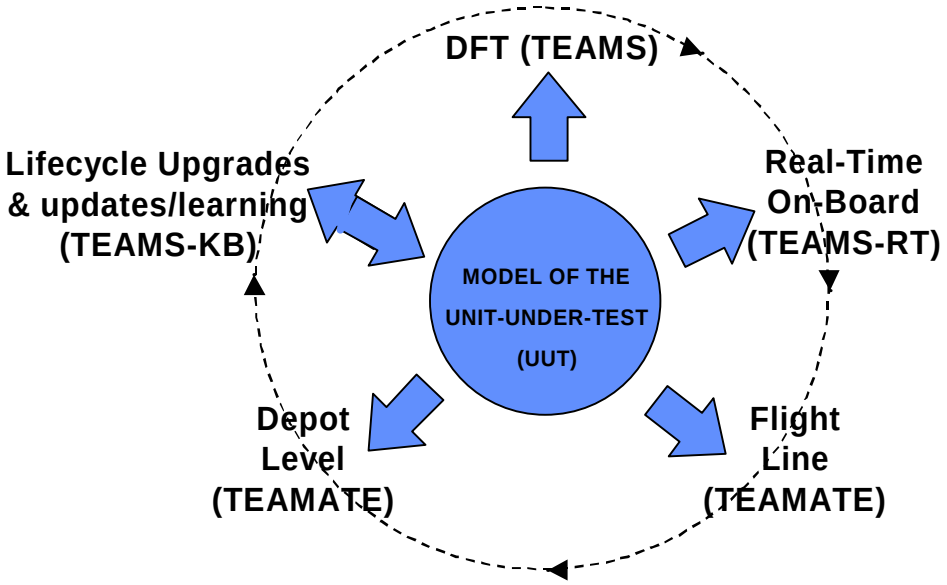

Figure 1: QSI's Model based Integrated Diagnosis Toolset

All of these tools utilize a common model of the system, wherein information about failure sources, tests and monitoring points, redundancy and system modes are captured in colored directed graph models known as multisignal models [1]. In simple terms, these models enable the inference engine to interpret test results by answering these questions: given a test T1, which components can cause it to fail; or, if I want to check the health of component C1, which tests can observe it. Such models may be automatically generated via fault simulation or developed in the TEAMS™ graphical user interface based on engineering understanding of the system or legacy data captured in FMECA reports, fault trees, CAD data, and technical documentation. These models are modular and hierarchical, closely related to the structure, and capture system specifications and test capabilities in simple graphical terms, and are therefore, easy to create and

maintain. Yet, they are rich enough to capture redundancies in fault-tolerant system, modes of operations in dynamic systems, setups, instrumentation, skills required to perform tests, and level of maintenance (i.e., Onboard, Flight Line, Depot Level). Use of the same model across all disciplines and maintenance phases ensures efficient and seamless transfer of diagnosis knowledge, avoids duplication of effort, and prevents any expectation gap between analysis and implementation. Thus, they implement a comprehensive and integrated life-cycle view of diagnosis and maintenance in sharp contrast to the fragmented approach in practice today. Further, since TEAMS™ uses the same model for testability analysis as TEAMS-RT™ (for onboard monitoring) and TEAMATE™ (for guided troubleshooting), the results predicted in the design stage are achieved in actual operations.

Current research at QSI is focussed on making this integrated diagnosis toolset accessible over the network so that systems can be remotely monitored and diagnosed. For many systems, such a solution is necessity. For example, the Space Shuttle and the International Space Station (ISS) rely on elaborate ground support systems for monitoring and management of system health. NASA mission control utilizes a highly trained team of engineers to provide ground support for all space missions. However, such elaborate ground support infrastructure was primarily designed to support missions of finite duration. For open-ended missions, such as that of the ISS, this is economically infeasible. A fast, scalable remote monitoring system is needed to continuously monitor the telemetry stream from the ISS, thereby reducing staffing requirements for around-the-clock monitoring. Further, this software system should be able to process the alarms, form a diagnosis, assess problem severity and its impact on mission, look up resolution procedures, and guided the engineer or astronaut through an optimized troubleshooting process, thereby improving response time to events, and providing just-in-time maintenance procedures and training to support staff.

At Qualtech Systems, we have developed Remote Diagnosis Server (RDS™) under a NASA Phase II SBIR that can support multiple simultaneous diagnostic and maintenance sessions from a variety of remote systems. Clients can connect to RDS™ over networks (wired, wireless, dialup connections etc.) and get health assessment and intelligent troubleshooting procedures over a web browser. The solution scales easily to hundreds of sessions in any modern workstation or server. In this paper we present the various design considerations and architecture behind our RDS™ solution.

2 THE REMOTE DIAGNOSIS SERVER

2.1 RDS™ Design Requirements

When designing the RDS™ architecture, the following considerations were of utmost importance to us:

1. Preserve investment in current TEAMS™ toolset: The TEAMS™ toolset represent years of research, development and validation by QSI. We wanted to preserve our investment in these tools and avoid rewriting and revalidating any of the code and algorithms. Further, we wanted the new architecture to make the runtime reasoners (i.e., TEAMS-RT™ and TEAMATE™) network accessible, while preserving their existing standalone and embedded mode of development.

2. Scalability: The resultant solution must be scaleable in the true sense of the word, i.e., it must be able to scale from small, embedded computers to massive servers. For example, existing applications of TEAMS-RT™ required that we be able to embed TEAMS-RT™ in a 75MHz Pentium computer to monitor a system with approximately one thousand sensors. In the server context, we would like to monitor hundred such systems in a small server, and thousands of such systems in a large server. Likewise, TEAMATE™ needs to be fully functional in a single user laptop environment, or in the context of a network server with hundreds of users accessing guided troubleshooting services over wireless networks using thin clients.
3. Transport neutral: RDS™ should operate in any message-passing network environments. Examples of message passing systems include 1553 bus systems, loosely coupled networks relying on sporadic satellite communications, the Internet, and local area networks. All these networks have one thing in common – they can send and receive messages. RDS™ was to be set up as an asynchronous system, where it receives messages, and responds to them. The transport layer is responsible for reliable message delivery, without any guarantee of throughput or latency.
4. Cross Platform Interoperability: RDS™ would be a client server solution where a centrally located server would monitor hundreds of clients running in a wide variety of hardware and operating systems. Thus, we should not make any assumptions about the client operating system, except for some minimal conformance to RDS™ specifications. Further, any software we develop for client environments must be portable, and hence written in ANSI C.
5. Service Provisioning: RDS™ would be monitoring multiple systems at the same time. However, we did not want any one client overrunning the server with too much sensor data. This was motivated by possibility of poorly designed chatty clients and denial-of-service attacks prevalent in the Internet. We took this criteria a step further and decided that the response time of RDS™ should be constant over its operating range so that service provided to any client is unaffected by the total number of clients in the system.
6. Online and Interactive services: RDS™ embodies two kinds of reasoning services. The TEAMS-RT™ implements online monitoring, where incoming data should be buffered and acknowledged quickly so that the onboard systems don't have to wait for RDS™ to process the data. This would be the non-blocking mode of operation. However, guided troubleshooting sessions, such as using TEAMATE™, are interactive, and the user waits (or blocks) until RDS™ provides the next step. Both forms of handshake and flow control had to be implemented in RDS™.
7. Load Balancing and Distribution: This is an extension of the scalability problem where if the computational requirements of RDS™ exceed the capacity of a single computer, it must be able to leverage multiple loosely coupled systems in a server farm or tightly coupled cluster. In such a configuration, distribution of clients across the multiple computers to alleviate bottlenecks is essential.
8. Usage tracking, logging, and reporting: RDS™ must be able to track each client session and support management and reporting functions, and archive health status information. Thus, a supervisor should be able to review the health status of the fleet, while technicians work on individual aircraft. This also required that users be able to monitor live sessions, thereby requiring multiple processes to attach to the same client data.

9. Leveraging existing technology and standards: We used established standards such as DNS, LDAP, XML, http, and TCP/IP whenever appropriate.
10. Security: RDS™, being a networked application, must be able to operate inside corporate firewalls and use standard methods for authentication and access control. For example, clients can be authenticated by checking credentials against an LDAP server, and by performing reverse DNS lookups.

2.2 Review of Existing Technologies

We began by reviewing existing server architectures and COTS development environments. The CORBA (Common Object Request Broker Architecture) [3] architecture appeared promising since it was ideally suited to network enabling of legacy applications. CORBA is also scalable, supports cross platform and cross-language interoperability, and can be implemented in distributed computer systems. However, it assumes a TCP/IP network, and is typically too bloated for embedded and low-ended systems. Nevertheless, we studied the CORBA architecture and drew inspiration from its implementation techniques.

CORBA is a specific method of remote procedure execution, where a client can access a service from a server, possibly running on a remote computer. In simple terms, the CORBA implementations provide *middleware* so that a client can access a service the same way it would access functions in a shared library. However, transparent to the end user, this *function call* is converted to a message string, which is then routed through one or more Object Request Broker (ORB) to the server providing the service, where the relevant *function* is exercised. The Java RMI (Remote Method Invocation) [4] implements similar mechanism, albeit without intermediate ORBs, and requires Java based client and server programs. RPCs (Remote Procedure Call) in Unix and NT [5] are OS-specific, but allow direct remote invocations.

2.3 The RDS™ Broker-Agent Architecture

The RDS™ framework (see Figure 2) is inspired by the CORBA [3] in that it allows client programs to remotely access diagnosis services over a network, and that there is a central computer or broker that matches the clients to the appropriate service provider. Similar to CORBA, all data is encapsulated in "strings", or "serialized" (see Figure 3), to enable the clients to invoke RDS™ services. We also borrowed concepts from shared memory architecture [6] and messaging protocols such as the Tooltalk protocol [7], to add functionality for message buffering, queuing and dispatching. In addition, we implemented concepts of "Handler" and "Observer" from the Tooltalk protocol, so that we could implement supervisory or reporting functions on top of normal monitoring and diagnosis services. Modeling of the RDS™ broker after the CORBA architecture also helped us preserve our existing TEAMS-RT™ and TEAMATE™ investment, while offering interoperability in a heterogeneous network, and leverage open standards like TCP/IP, DNS, LPAP etc.

However, the broker implemented in the RDS™ architecture is quite different from the ORBs in CORBA and offers some unique capabilities.

- It implements both blocking (i.e., the client call blocks and waits for the server to complete execution, as in CORBA) and non-blocking (i.e., the broker acknowledges the receipt of message and releases the client call, as in message queues and Tooltalk) mechanisms. The RDS™ broker automatically selects the mode appropriate for the service (i.e., non-blocking

mode for online monitoring, and blocking mode for guided troubleshooting). This helps us satisfy the sixth requirement outlined in section 2.1.

- It is customized to our application, and therefore a much more compact implementation than CORBA. The Broker code is currently less than ten thousand lines of code, and yet implements functions for session management, flow control, load distribution, usage tracking, logging, etc., which are often lacking in all but the high-end CORBA implementations. This helps us scale down to small embedded applications with relative ease (requirement 2 in section 2.1)
- It is a hybrid model, where serialized messages are buffered in a shared memory message queue instead of being delivered directly to another ORB or the server. This is because message passing and remote method invocation systems usually sequester data structures behind a centralized manager process; therefore, any process that wants to manipulate the structures will have to wait its turn and ask the manager to perform the operation on its behalf. In other words, multiple processes can't truly access a data structure simultaneously in these conventional systems. A directly accessible message pool allows multiple services to act on the same message. This facilitates use of specialized programs for reporting, logging, and diagnosis, all acting on the same message, in the RDS™ framework (requirement 8 in section 2.1).
- It performs flow control to ensure that any particular client does not hog disproportionate amount of server resource. For non-blocking clients, RDS™ buffers the incoming messages in a queue and throttles the client whenever the queue limits are reached. For blocking clients, RDS™ enforces a response time of one second even on a lightly loaded server. Thus the user always observes a one-second processing time, irrespective of the number of users in the system, as long as the server is at or under capacity. This helps us satisfy our key requirement for service provisioning stated in the section 2.1.
- It uses a broker-agent architecture where agents monitor the shared memory buffer managed by the broker for actions, rather than the broker notifying handlers when new requests come in. This concept is similar to the JavaSpaces methodology [8], where processes don't communicate directly, but instead coordinate their activities by exchanging objects through a space, or shared memory. These processes, or agents, are task oriented, and are used to incorporate new or existing services (e.g., TEAMS-RT™ and TEAMATE™ reasoning services) into the RDS™ framework.

Perhaps the most unique feature of the RDS™ architecture is this broker-agent architecture that merits further discussion. As an example, consider an online auction system that brings buyers and sellers of goods and services together. Suppose you, as a potential buyer, describe the item (such as a car) you'd like to buy and the price you're willing to pay, wrap the information in an message, and send it to broker, who puts it in the message pool. At the same time, potential sellers continually monitor the message pool for the arrival of wanted-to-buy entries that match items in their inventory. For example, Mazda dealers monitor the message pool for entries that describe Mazdas, while used-car dealers monitor the message pool for all used-car requests. When a matching request is found and read, a potential seller writes a bid entry into the message pool, stating an offering price. As a potential buyer, you continually monitor the message pool for bids on your outstanding requests, and, when you find one that's acceptable, you remove the bids and contact the seller (possibly through another message). In the above analogy, the buyers

and sellers correspond to the clients and agents respectively, while the broker facilitates exchange of messages by managing the integrity of the message queue, cleaning up expired or processed messages and allocating buffers to hold the messages. This approach scales naturally: it works the same way whether there are 10 buyers available or 1,000. The approach also provides natural load balancing, since each agent picks up exactly as much work as it can handle in a given time, with slow agents doing less work and fast agents doing more.

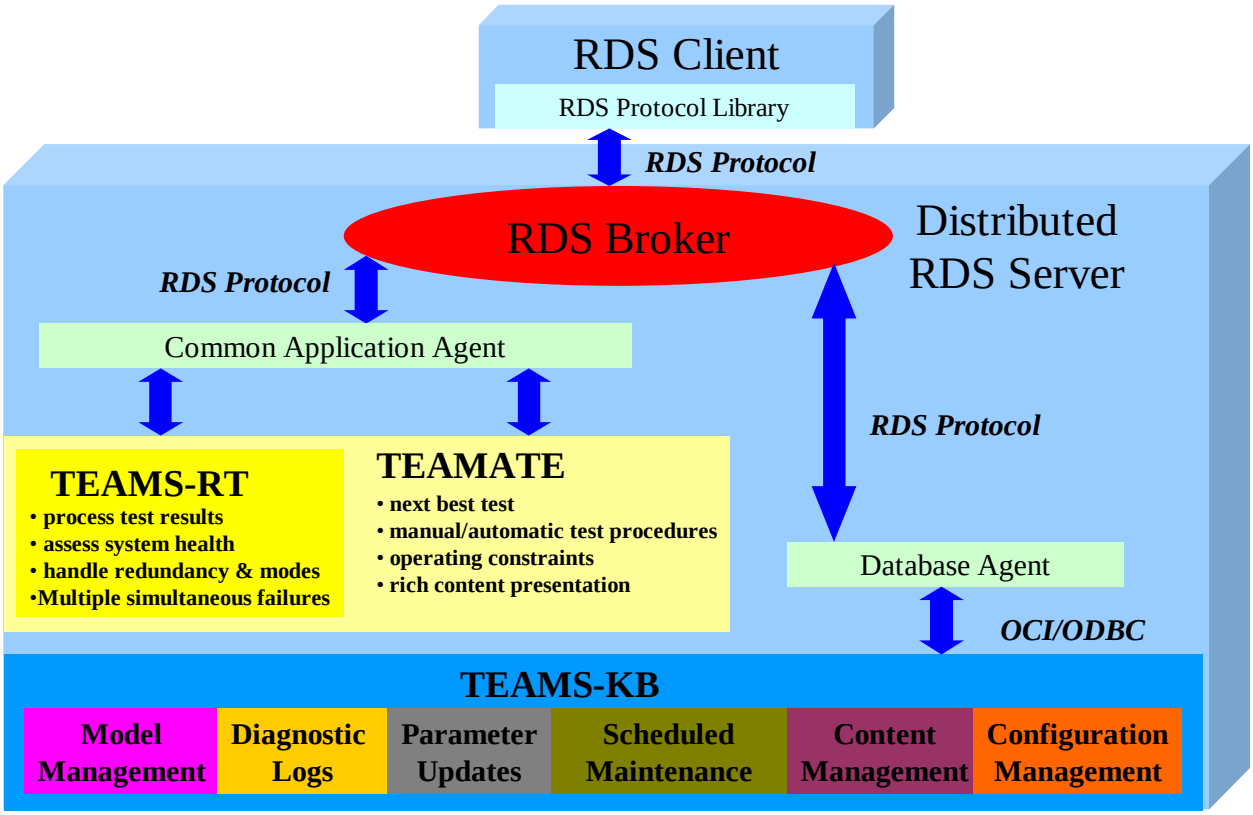

Figure 2: The Remote Diagnosis Server Framework

The RDS™ architecture is built around the philosophy that the scalability of the software will be derived from the Broker and, therefore, the Broker will have to be very efficient and lightweight. Further, it should be possible to add new functionality without increasing the complexity of the Broker. Consequently, the Broker is service-neutral by design. While it manages the constituent services and sessions, it has no knowledge of the underlying mechanisms or data dependency of the individual services. The constituent services of the RDS™ are implemented by the appropriate service providers (e.g., TEAMS-RT™ and TEAMATE™), as abstracted to the Broker by the corresponding agent. The beauty of the architecture (Figure 2) is that all tasks are performed by a multitude of agents, each with a specialized function, while the broker performs housekeeping functions, such as session and message pool management, and garbage collection. The resultant solution is also more manageable and extensible compared to alternate monolithic architectures, and scales efficiently from desktop computers to servers with dozens of processors. It also supports upwards of 300 concurrent clients in modest workgroup server configurations.

The essential constructs of the RDS™ framework have been described in earlier papers [9,10] and are not repeated here. More information and PDF versions of the papers are available in http://www.teamqsi.com/rds.

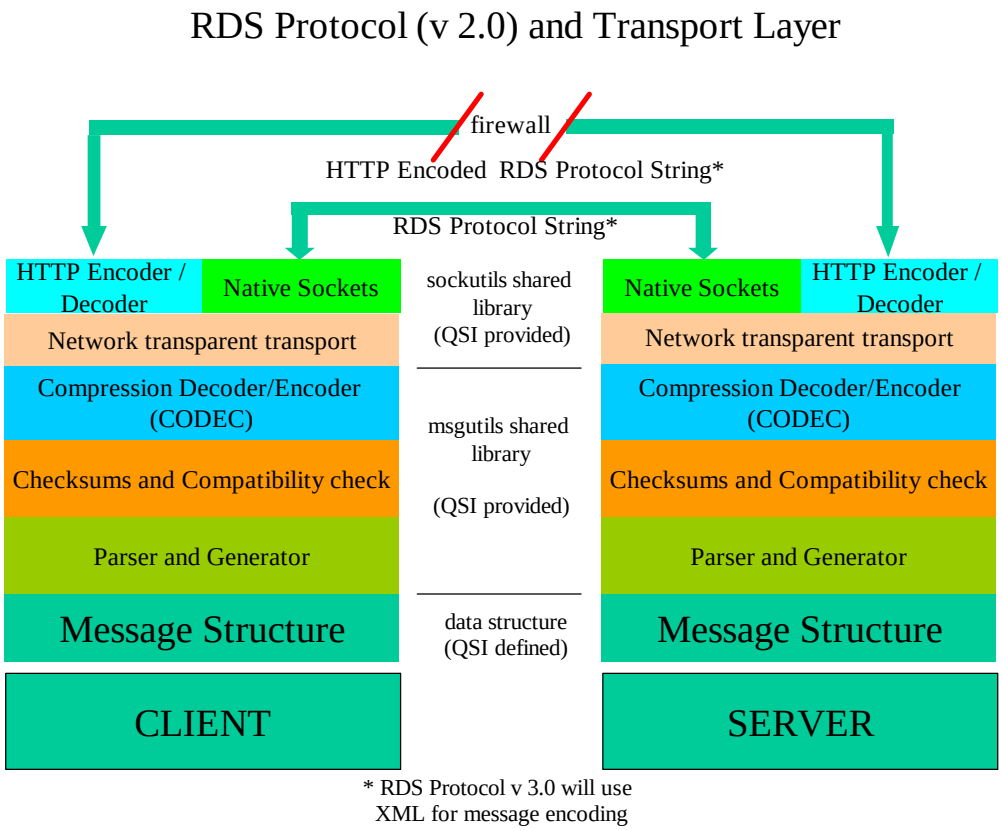

Figure 3: RDS™ Protocol and Transport Library – simplifying low cost remote client development.

2.4 Performance Results

We ran extensive simulations to test the scalability of our RDS™ solution. Our current computational resource restricted us to using up to 500 concurrent clients, although we feel very confident that the architecture will scale to thousands of clients in larger computers. We simulated problems of size 100, 1000, and 10,000 failure sources. For problems of size 100, we could easily simulate 500 concurrent clients, whereas for 10,000-failure source system, we could only simulate 8 concurrent sessions. This is because the complexity of the problem grows linearly with the number of failure sources or number of tests. In all cases, the processing time per client stayed constant, independent of the number of clients, which satisfies one of our major design criteria.

In the phase II SBIR, we also modeled a section of the 1553-bus system of the ISS [11], a highly redundant, re-configurable, dynamic, fault-tolerant system. We used this model to verify the accuracy of our diagnosis algorithms and to test its performance. Two sets of tests were performed. In the first set, we seeded random faults in our sensor_agent test data generation

program [9,10], and generated data that emulates the observed test data from the telemetry stream. We then uploaded this test data to RDS™ for diagnosis, and compared the RDS™ generated diagnosis against the seeded faults. In the second set, we set up about ten concurrent test cases where the data was continually uploaded every 1 second interval, and the seeded fault randomly changed at irregular intervals. We evaluated the diagnosis accuracy on both cases, and found it closely matches the detection and isolation performance predicted by testability analysis. The second test case also measures the throughput performance of the RDS™ server. On a SUN workgroup server (E250) with two 400 MHz UltraSPARC II processors, the CPU utilization was under 10% during test set 2, indicating this configuration could monitor approximately a hundred similar systems onboard the ISS without any difficulty. We verified this claim by running 100 concurrent interactive diagnosis sessions with RDS™ and TEAMATE™ without any computational bottleneck. Further, the processor and memory utilization scaled almost linearly with the number of sessions, while the response time remained below the 1-second data interval. Based on these results, and simulation tests on many other models, we feel confident that our RDS™ solution can perform effective concurrent monitoring and diagnosis of tens and hundreds of ISS systems in real time.

3 CONCLUSIONS

In this paper, we presented our experience in developing a telediagnosis architecture for monitoring, diagnosis, and health management of remote systems. This effort was initially motivated by the need for a remote monitoring solution for the International Space Station. The resulting server, RDS™, can simultaneously support hundreds of client sessions over standard Internet protocols such as TCP/IP and http. The resultant architecture is now in patent pending status.

QSI will launch its Remote Diagnosis Server product in summer of this year. There is considerable interest from NASA and other leading aerospace companies that lead us to believe that the RDS™ solution will soon be applied to the dignosis of a variety of remote systems. For example, Honeywell is using our RDS™ software and TEAMS™ models to develop a comprehensive demonstration for remote monitoring of the space station based on telemetry data. In an internal project, they are setting up software to mimic real-time extraction of sensor and test data from the telemetry stream, and using RDS™ for health assessment and diagnosis. For the demonstration, Honeywell has selected a portion of the power distribution system identified as LAAFT-2B Power Distribution Assembly. Other applications include monitoring of engine health and a comprehensive wire integrity program involving routine and unscheduled maintenance of wiring systems of aging aircraft.

The development of RDS™ is a major milestone in our plan for commercializing integrated system design, diagnostic and prognostic tools. Our integrated toolset helps achieve lower life-cycle costs by addressing reliability, testability and maintainability issues: failure analysis, design for testability, automated testing, interactive diagnosis, and real-time system health monitoring. While many of our competitors offer products in the areas of integrated diagnosis, most lack a real-time diagnosis engine, and none have a networked diagnosis server capability. Until now, real-time diagnosis and prognosis have been available to a selected few multi-million dollar applications. The remarkable aspect of this technology is that it is accessible over Internet and modems, making real-time diagnosis universally accessible! This is a key discriminating

factor that will enable us to reach beyond the niche market of integrated diagnosis, and tap into consumer applications and e-business.

For example, the modern automobile has enough sensors to detect the slightest performance problem. The engine computer(s) monitor fuel mixture and ignition system for optimal fuel efficiency, drive-train computer(s) monitor the grade of the road, torque and acceleration to select the correct gear, and antilock brake systems detect wheel lock ups and dynamically adjusting for brake wear. Some high-end models already come equipped with communication links (e.g., OnStarTM by Cadillac) that can report mishaps, e.g., an accident causing airbag deployment, to a central monitoring station. In a few years, such features will be available on all cars. Presently, such communication links are offered primarily as a safety net, or as a link to customer and concierge services. However, they can easily be adapted to transmit onboard data to a RDS™ service where car troubles can be quickly diagnosed. It is therefore conceivable that soon, the driver of a stalled car will be able to get a prompt diagnosis using RDS™ service, and AAA would dispatch roadside assistance with the exact spare part required to fix the problem. The applications of RDS™ are not limited to the automobile. Remote health monitoring of home-care patients and battlefield soldiers are two of the more promising applications. Modern high rise buildings consist of elevators, escalators, heating and ventilation systems etc. that also need to be monitored around the clock. Utilizing RDS™, a central facility could monitor entire cities of high-rise buildings from one central location.

RDS™ is an essential piece of technology that makes such applications feasible.

ACKNOWLEDGMENTS

This research has been sponsored by a Phase II SBIR award from NASA Ames Research Center (NAS2-99049).

REFERENCES

1. Somnath Deb, Krishna R. Pattipati, M. Shakeri, V. Raghavan, and R. Shrestha, “Multi-signal Flow Graphs: A Novel Approach for System Testability Analysis and Fault Diagnosis.” In. Proc. 1994 IEEE AUTOTEST Conference, Anaheim, CA.
2. Somnath Deb, Krishna R. Pattipati, and R. Shrestha, “QSI’s Integrated Diagnostics Toolset”, in Proc. IEEE Autotestcon 1997, Anaheim, CA, pp 408-421.
3. “CORBA Fundamentals and Programming”, Ed. Jon Siegel, John Wiley & Sons, N.Y., 1996. Also see http://www.omg.org/corba.
4. “Java.rmi : The Remote Method Invocation Guide” by Esmond Pitt, Kathleen McNiff, Kathy McNiff (Addison-Wesley, 2001).
5. “Power Programming With RPC” by John Bloomer (O'Reilly & Associates, 1992)
6. “Distributed Shared Memory : Concepts and Systems” by Jelica Protic (Editor), Milo Tomasevic (Editor), Veljko Milutinovic (Editor), IEEE Computer Society, 1997.
7. “Common Desktop Environment 1.0 Tooltalk Messaging Overview” (Common Desktop Environment 1.0), CDE Documentation Group, Addison-Wesley Pub Company, N.Y., 1995.
8. “JavaSpaces Principles, Patterns, and Practice”, Eric Freeman, Susanne Hupfer, and Ken Arnold (Addison-Wesley, 1999).

9. S. Deb, S. Ghoshal, V. N. Malepati, and D. L. Kleinman, “Tele-diagnosis: Remote monitoring of large-scale systems”, in Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, March 18-15, 2000.
10. S. Deb, S. Ghoshal, V. N. Malepati, and K. Cavanaugh, “Remote Diagnosis Server”, in Proceedings of the IEEE Digital Avionics Systems Conference, Philadelphia, PA, Oct. 7-13, 2000
11. S. Deb, C. Domagala, S. Ghoshal, A. Patterson-Hine, and Ri. Alena, ” Remote Diagnosis of the International Space Station utilizing Telemetry Data”, in Proceedings of the SPIE Aerosense Conference, Orlando, FL, April 16-20, 2001.