

# JEDSI: Java Enhanced Distributed System Instrumentation

*Barbara A. Shapter and Douglas G. Crowe*

**J**ava Enhanced Distributed System Instrumentation (JEDSI) makes it possible to capture performance or diagnostic data, or both, from a distributed software system and to analyze the data and display the results while the software system is running. The JEDSI software is designed to have minimal impact on the instrumented system, to be flexible in its application, to be extensible for new analyses, and to provide a variety of customizable displays. This real-time concurrent display of software performance becomes a critical tool in the development, test, tuning, and support of distributed application systems. The JEDSI tool provides a way to instrument an application so that its performance can be observed while it is in use.

## INTRODUCTION

One of the challenges in developing large distributed software systems is measuring their performance. Historically, a common way to measure performance or analyze behavior was to capture data from the system by writing the data to a file for later analysis. This approach is cumbersome and can require multiple, time-consuming test runs to fully explore a system's performance characteristics. In addition, owing to sometimes extensive coding overhead and potentially detrimental impacts on performance, redefining the data collected during series tests can be very difficult.

The High Performance Distributed Computing (HiPer-D) project at APL required a tool that could provide a "live" view of the performance of its combat system simulation. Earlier versions of HiPer-D used the JEWEL instrumentation tool from the German National Research Center for Computer Science.<sup>1</sup> From the Laboratory's experience using JEWEL with HiPer-D came

the decision to develop a more flexible software instrumentation tool. The primary functional requirements for this tool were

- Collection of performance data from processes in a distributed system with minimal impact on performance of the system under test
- Evaluation of collected data in near-real time, with provision for easy extension and the addition of customizable analysis components
- Dynamically selectable, configurable, real-time displays
- Ease of learning and use
- Usefulness in a heterogeneous distributed hardware environment
- Adaptability to new problems

The prototype Java Enhanced Distributed System Instrumentation (JEDSI) was developed in 1998. These requirements motivated its architecture and the

selection of Java for use in the central data collection and dynamic display components.

The process of capturing data at test points in a software system is analogous to putting probes in a physical system. JEDSI's "probes" are software function calls that pass information from the system under test to JEDSI. Instrumenting a software system with JEDSI is the process of identifying the data to be captured and inserting the JEDSI function calls needed to capture those data into the system under test. Software systems that have undergone this process are called instrumented systems, and the individual data collection function calls are referred to as test points.

Before the development of JEDSI, one of the issues associated with any software instrumentation system was the number of test points that could be inserted into the system under test. Each data value extracted from a test point had to be removed, which required processing cycles that were then not available to the system. The need to limit the data extracted, and thus the required processor overhead, has been a limiting factor in the number of test points traditionally placed in systems under test. In addition, the overhead associated with extracting data tended to lead developers to remove those test points after initial development, thus eliminating a valuable source of information that could be useful in pre-deployment integration efforts and post-deployment problem analysis.

JEDSI addresses these issues by assuming that all embedded test points are turned off. No data are transferred from these test points. Test points are turned on only when a user of JEDSI needs the data. This approach removes the concern with the volume of data being transferred when large numbers of test points are left embedded in the system. JEDSI test points can be used freely to make available as much information as might be needed. Test points also can be safely left in code for use in integration testing and post-deployment diagnostics.

## JEDSI ARCHITECTURE

JEDSI provides a small, flexible Application Programming Interface (API) for use in adding test points to software systems that are to be instrumented. JEDSI also contains a set of portable components for data collection, analysis, and display that can run on one or more platforms. The term "distributed" in JEDSI's name applies both to the tool's ability to process data from a distributed system and to the ability to distribute the components of JEDSI itself. The JEDSI architecture allows

- Simultaneous, real-time collection of data from multiple processes under test on a single host platform
- Simultaneous, real-time collection of data from multiple host platforms that are running one or more systems under test

- Collection of data for analysis and/or distribution at a central point
- The ability to provide both raw and analyzed data to multiple JEDSI clients on multiple platforms

Figure 1 shows the main components of JEDSI:

- The JEDSI API, which is the user's programming interface. It defines the functions needed to pass information from the program of the system under test to the JEDSI API library. This library is linked with each program and becomes part of that program.
- A JEDSI Receptor, which runs on each computer platform that hosts the program of a system under test. The Receptor receives data from all instrumented programs of the system on that machine and transmits the data to the JEDSI Collector. JEDSI Receptors and the API library are available for the Solaris, HP UX, Linux, VxWorks, and Windows 2000/NT environments.
- The JEDSI Collector, which gathers the data from instrumented test points from all the Receptors in the system. These data are analyzed (if needed) and passed on to the JEDSI client processes.
- The JEDSI Test Control display, which is the data display client of the Collector. One or more client processes can receive the data collected by the JEDSI Receptor and/or the results of the analysis of the instrumentation data.

Figure 1 shows three possible client processes: the JEDSI Test Control display application, a Web application, and a process that writes data to a log. The API, Receptor, Collector, and Test Control are discussed in more detail in the following sections.

## JEDSI Application Programming Interface

The JEDSI API<sup>2</sup> provides a simple and flexible programming interface for capturing data from the software system. Two function calls are used most often—one that performs initialization and one that is used to pass data into JEDSI. The initialization function initializes the interprocess communications between the instrumented process and the local Receptor. The data capture function sends data to JEDSI using a simple, flexible syntax that accepts any number of arguments. Each data capture call passes the information supplied to the JEDSI Receptor, along with the time of the event, the identifier for the test point, the host network address, and the process ID. The API is implemented in C and is suitable for reference from C or C++ programs. Other languages that have the ability to call C libraries (most languages have this capability) may also be used with JEDSI.

## JEDSI Receptor

The JEDSI API library that is linked into all instrumented programs of systems under test sends the data it

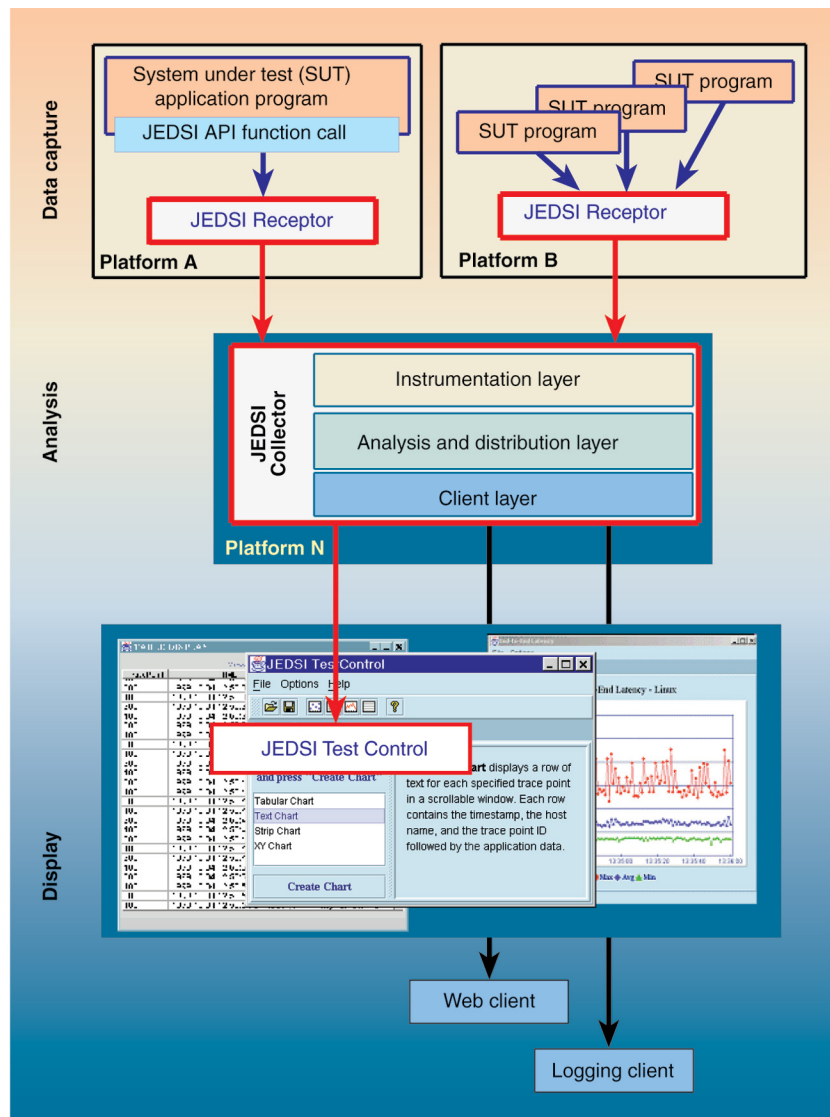


Figure 1. JEDSI component architecture.

collects to the Receptor. The JEDSI Receptor receives the data from all instrumented application processes on its platform. Currently, one Receptor can support up to 20 instrumented processes. The Receptor forwards all the data it receives from the various API libraries to the JEDSI Collector using a platform-neutral protocol. This means that the Receptor and the Collector need not reside on the same type of machine.

A key design requirement for the JEDSI Receptor and API library is to permit the collection of the application’s run-time characteristics and data with only minimal impact on the performance of the instrumented application. Consequently, the Receptor and API are written in C and use shared memory to obtain the performance data captured by the API calls.

All instrumented processes on a given platform communicate their data to a common Receptor. However,

more than one Receptor can be run on a given host. This might be done to isolate event data from two different systems hosted on the same computer.

A distributed system can contain a large number of JEDSI instrumentation test points. However, the data from all of the test points may not necessarily be desired at run time. For example, different subsets of test points may reflect different, mutually exclusive test scenarios. To explore a particular scenario, only the pertinent test points should be enabled. The Receptor allows this selectivity in controlling test points at run time by dynamically enabling or disabling the test point state in response to control messages from the Collector. A test point is enabled if at least one client of the Collector is a consumer of its data.

### JEDSI Collector

The Collector is a Java application that receives data from all Receptors, performs any required analysis, and communicates results to client processes. The Collector consists of Java components that run in several different threads of control within the Collector. The Collector runs on any host, but it is recommended that the Collector be run on a different host from that used by the processes of the system under test to eliminate the

Collector’s impact on the system’s performance.

The communications interface between the Receptor and Collector is implemented using standard Transmission Control Protocol/Internet Protocol (TCP/IP). The use of TCP/IP as a communications protocol means that the Collector and its Receptor host processors can be connected by any modern communications mechanism. Ethernet, Fast Ethernet, and ATM can all be used. The Collector can be located on the same local area network or at a remote location connected using a wide area network.

The messages exchanged between the Collector and Receptor consist of the following:

- Initialization messages that initialize communications between the Receptor and Collector. (The information transmitted includes the JEDSI version

number of the Receptor that is sending messages, which allows the Collector to operate with different versions of the Receptor simultaneously.)

- Data messages that contain the information from the system under test collected by the Receptor
- Receptor host environment information messages, such as a message indicating that an instrumented process has started or ended
- Test point control messages from the Collector that enable or disable test points in the instrumented process

The Collector's design is a three-tiered architecture (Fig. 1). The first tier is the instrumentation layer, which contains components that support communications with the Receptors. These components detect new connections and create a separate thread of control to handle all message traffic between the Collector and each new Receptor. The second tier, the analysis and distribution layer, contains queues, control components, and components that perform analysis of the captured system data. The third tier, the client layer, consists of components that support communications with JEDSI clients. JEDSI clients are the consumers of the data collected by the Receptors and/or the results of any analysis performed on these data within the Collector. Clients connect to the Collector using separate TCP/IP connections, one for each client. Each client connection with the Collector has a separate thread of execution dedicated to serving data to that client. When the client connects, it provides parameters to the Collector that identify the data from the Receptor that it needs. Parameters are also included that identify any data analysis required by the Collector.

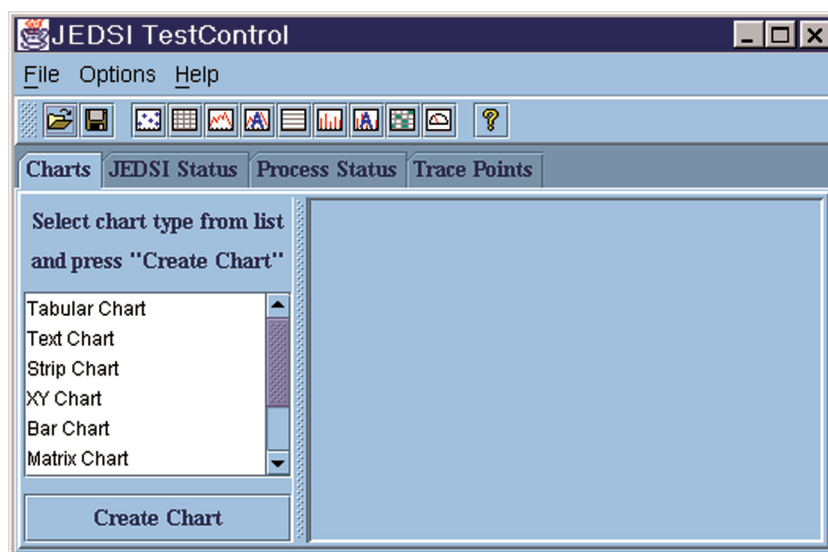
Data analysis is implemented within the Collector using separate components known as Evaluators. A separate Evaluator is started with each client connection. The Evaluator extracts the data specified by the client from the stream of data coming from the Receptors. It performs whatever processing is required on those data and forwards the results to the client. Several general-purpose Evaluators are supplied with JEDSI, including a pass-through Evaluator that simply forwards data directly to the client without change. Users can also easily develop their own Evaluator components. JEDSI's use of Java is such that user-developed Evaluators can easily be incorporated without changing any released JEDSI source code, thus preventing configuration

management conflicts between user-developed code and the JEDSI baseline code.

The Collector also provides central control of test point on/off state information. When a client connects to the Collector requesting specific data, the Collector compares the data requested with its list of test points that have already been turned on by previous requests from other clients. If the test point is not enabled, a message is sent to the appropriate Receptors to turn on that test point. Data then begin to flow from the system to the Receptor, through the Collector, and out to the client. The Collector monitors the status of all of its clients, waiting for a client to disconnect. When all clients that have requested data from a particular test point have disconnected, the Collector sends a message to the Receptors to disable that test point. This prevents the flow of unneeded information from that test point, thus conserving communications bandwidth.

## JEDSI Test Control

Test Control is a Collector client and serves as the display component of JEDSI. Like the Collector, it is a multithreaded Java application that can be run on any platform that supports Java. Test Control provides a graphical user interface to its collection of display tools from its main window (Fig. 2). From this window, a user can choose a chart for use in connecting to the Collector data stream and observing the status of the JEDSI components. By default, raw data collected by the Receptors are displayed in Test Control charts without additional filtering or processing. A user selects a chart or clicks on a chart icon on the tool bar to start up a new data display.



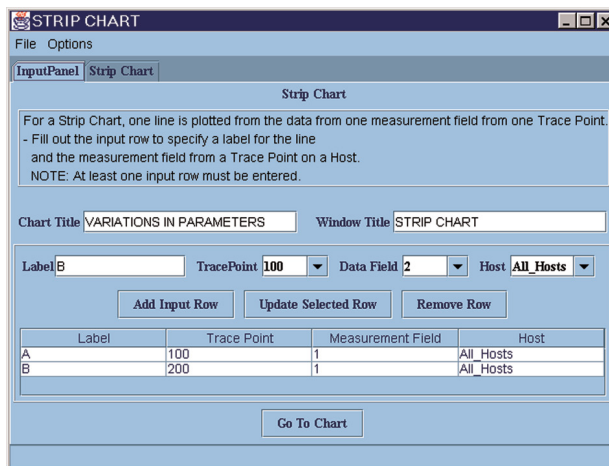
**Figure 2.** The JEDSI Test Control main window provides tools to create, modify, save, or retrieve charts, as well as tools to monitor the state of the JEDSI components and instrumented processes.

Each chart is displayed in a separate window containing tabs for both the chart and an input panel. The input panel provides a place for the user to define chart parameters such as the test points that will be used as the data sources, the window name, the chart name, etc. The window for a strip chart is shown in Fig. 3 with the Input Panel tab selected. There are text fields with drop-down lists of known test points, data fields, and system hosts for input. Furthermore, all parameters entered are available for review and revision at any time. The input parameters of each chart can be changed on the fly by clicking on the Input Panel tab and changing the parameters. Clicking on the Chart tab or on the Create Chart button applies the changes to the chart. Each time a chart window is opened, a separate client connection is made to the Collector that requests data from the test point(s) specified in the input panel.

A group of seven JEDSI Test Control chart windows is shown in Fig. 4. The data displays appear in separate windows, so several different displays can be shown simultaneously and arranged and sized as desired. At present, data can be displayed in a table or dynamic matrix, as text, or as a strip chart, bar chart, xy chart, or gauge. Test Control includes other features as well:

- On-line help
- A detachable toolbar of shortcuts to charts and other capabilities
- The ability to save and restore chart configurations
- Chart printing
- Saving data to a file
- Customization of chart appearance
- Status displays of the connectivity state of each JEDSI component and each instrumented process

Test Control contains an innovative specialization of a standard table of data. This is the dynamic matrix chart. A dynamic matrix chart is a matrix whose row and column headers, as well as its cells, are data driven. A traditional table has a fixed number of columns, defined at startup, with static headers that are defined as constants. A dynamic matrix chart, on the other hand, displays data as a growing list of rows and columns. Each update potentially adds a new row or a new column. In a dynamic matrix chart, the size and shape of the matrix, as well as its content, depend on the data. The column and row titles are supplied by the data. If a new value



**Figure 3.** Typical JEDSI chart window showing the chart Input Panel tab. After the chart parameters are entered, clicking the Go To Chart button or the Chart tab displays a chart with data from these test (trace) points.

for a column or row title arrives, then a new column or row is added. Thus, the number of rows and columns in the matrix changes during the life of the chart. Any data message received can add a new row, add a new column, or change data in a cell of the matrix. The matrix chart



**Figure 4.** Seven JEDSI Test Control chart windows illustrating the variety of chart displays available.



is responsive to the data it receives, and its changes with time reflect the evolution in the behavior of the data it tracks.

## EXPERIENCES USING JEDSI

### HiPer-D

HiPer-D is a combat system simulation in which radar messages flow from one end of the system (the radar simulation) to the other end (track data consumers). The HiPer-D program has had many uses for JEDSI. One of the critical issues in the HiPer-D architecture is the time taken for messages to flow through the system. This time is the latency measurement. Initially, JEDSI was used to measure end-to-end message latencies.

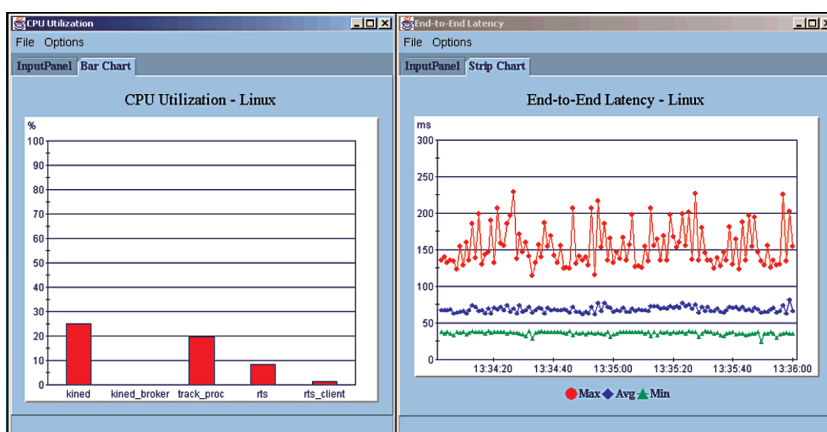
In analyzing the performance of HiPer-D components, it is useful to understand variations in the load that the processes place on the processors. To help assess this variation, a separate process was developed that periodically measured the central processing unit (CPU) utilization of various processes. This process was instrumented with JEDSI test points, and a dynamic bar chart was used to display the results. Figure 5 illustrates sample latency charts and processor load charts used in HiPer-D.

Part of the HiPer-D program involves the evaluation of commercial technologies for possible use in Navy systems. JEDSI was used to compare the performance of HiPer-D running in a Solaris environment with the performance of the same system running in a Linux environment. Figure 6 shows two of the early latency measurements made as part of this evaluation (two separate runs; note the time difference on the x axis of the strip chart). The significant difference in performance was unexpected. The network configuration used with this test consisted of a single switch that connected the Solaris and Linux systems. This switch implemented subnets that were used to prevent multicast traffic between the Solaris and Linux systems. In this environment the two systems (Linux and Solaris) were run simultaneously, and their latency behavior was observed using JEDSI. The Linux system was started

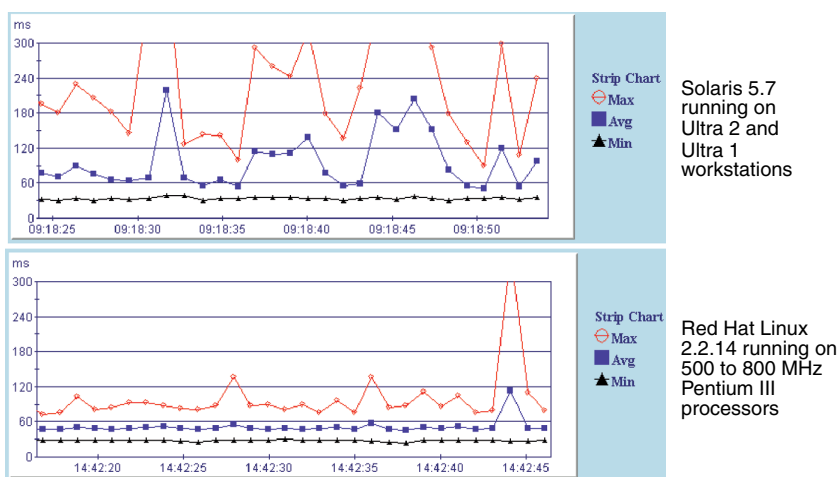
first. Then the Solaris system was started, and the JEDSI strip chart monitoring the Linux system performance immediately showed significant increases in latency. JEDSI showed that the two supposedly isolated systems were apparently interfering with each other. Further investigation revealed that the network switch was not blocking multicast traffic as intended. After consultation with network managers and reconfiguration of the switch, the experiment was repeated. JEDSI showed that the two systems no longer interfered with each other and that performance in the two environments was much closer, as was initially expected. In this situation, JEDSI helped uncover a network configuration problem that had previously gone undetected.

### Multiship Simulation

HiPer-D has been used in the Distributed Weapons Coordination (DWC) simulation effort in which a



**Figure 5.** JEDSI charts for HiPer-D showing simulation performance measures. The bar chart (left) indicates the CPU utilization of various HiPer-D processes. The graph (right) shows the average latency of messages as they flow through HiPer-D.



**Figure 6.** HiPer-D latency comparison between Solaris and Linux using JEDSI.

multiship simulation is used to test algorithms designed to coordinate the engagements of the multiple ships against a raid of Tactical Ballistic Missiles (TBMs). In the development of this simulation, JEDSI was used extensively to monitor both the engagement decisions made and the data used to make those decisions.

In the DWC simulation, each ship exchanges engagement status and interceptor inventory information in real time as a simulated TBM raid progresses. Each ship simulation sends engagement status information for each TBM in the raid every time the engagement status changes. Each ship in this simulation independently and periodically evaluates the engagement status data from all ships for each TBM and follows a set of rules to determine if it is the “preferred shooter,” i.e., the ship that should complete the engagement. Ships that are not the preferred shooter continue to process the engagement until interceptor launch time, when they suspend the launch if they are still not the selected shooter.

This real-time simulation exchanges a great deal of information among ships and makes a large number of decisions about that information. JEDSI was used to help visualize and analyze the data in this extremely dynamic real-time environment. Figure 7 is a snapshot of three of the JEDSI displays used. The top two charts are dynamic matrix charts. In these charts there is a row for each TBM (identified by track numbers 100 to 119) and a column for each ship (labeled LS1 to LS4) in the simulation. The left chart of Fig. 7 shows the rule

in the decision process last used for that TBM by that ship to decide if it should be the preferred shooter. Dark blue cells in the matrix indicate the ship selected as the shooter for that TBM.

The upper right chart of Fig. 7 shows the engagement status of each ship against each TBM as the coordinated engagement proceeds. The dark blue cells indicate the ship selected as the preferred shooter against the TBM, and a green cell would indicate that the ship had actually launched an interceptor against the TBM. Finally, the chart in the lower right of Fig. 7 is a matrix that shows the missile inventory for each ship, which is also a factor in the shooter selection process.

Each ship reevaluates its preferred shooter status against each TBM twice per second. In addition, engagement status information changes rapidly, especially in the earlier phases of each simulation. The result is a highly dynamic display. Initially, recording software was used to capture an electronic movie of the screen image as the simulation progressed, and this was played back at slower speeds to analyze the exact series of events. This method proved cumbersome, so a JEDSI strip chart was developed to capture a detailed timeline of the events that occurred during the simulated raid. An example is shown in Fig. 8.

Each horizontal line in Fig. 8 represents a single TBM, and the four rows of marks below each line illustrate when events occurred for each ship simulation relative to that TBM. The events tracked were changes in engagement status and changes in the preferred shooter selection (by ship). This chart provided a sequential history of the information that was displayed in real time in Fig. 7. It was instrumental in developing an understanding of the sequence of events that occurred, and it helped in modifying the rules that the ships followed to determine the preferred shooter so as to maximize the ability of the four ships to handle the entire TBM raid.

### Live Test Monitoring

JEDSI has been used to monitor live data from Navy exercises. Live tactical data in the form of Link 16 J-series messages were received, and the resulting track picture was displayed. In this environment, JEDSI test points were inserted into the software reading the data stream. JEDSI Test Control was used to display strip charts of the parameters, such as altitude, for each track, dynamically adding and labeling new lines as new tracks were added. This provided a live, visual history of the data that would not have otherwise been available. JEDSI charts allowed operators monitoring the exercise the ability to see immediately which tracks represented the booster and when dual reporting conditions existed on the Link 16 network. Figure 9 shows the setup for such live test monitoring, with JEDSI charts shown on the center screen.

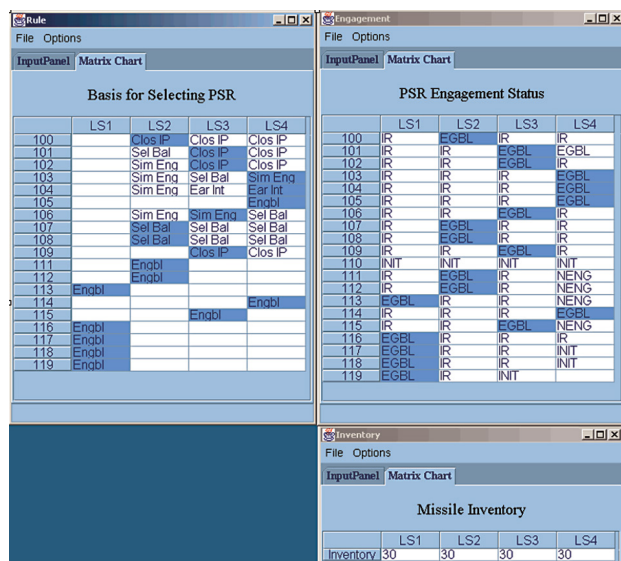


Figure 7. JEDSI matrix charts showing engagement status at a late stage in a multiship simulation of a TBM raid. Column headers reflect four ships; row headers are TBM track numbers. New rows and columns appear as new ships and threats appear in the simulation. The matrix chart is responsive to the data it receives, and its changes with time reflect the evolution in the behavior of the data it tracks.



**Figure 8.** JEDSI strip charts used as timelines to visualize individual events in the engagement decision process during a multiship raid simulation.

## POTENTIAL FUTURE USES

Lockheed Martin has developed JEDSI Receptors for inclusion in Aegis Baseline 6 Phase 3. This could lead to the use of JEDSI to monitor combat system performance in real time, as seen in Fig. 10. The figure illustrates another important possibility. The connection between JEDSI Test Control and the JEDSI Collector is a simple network connection, which could be implemented over a radio network. This would allow real-time monitoring of combat system software performance from remote locations.



**Figure 9.** JEDSI monitoring live exercise data.

## CONCLUSION

JEDSI provides the benefits of a live, real-time view into a distributed software system. It is a tool that is quickly learned and easy to use. JEDSI's simple instrumentation API favors flexible and fast implementation. Its Test Control tool provides simple, fast, and flexible displays. The JEDSI architecture encourages extension to new analysis needs, without requiring the instrumented system or any component of JEDSI to be rebuilt.

JEDSI is scalable and distributed by design. As the instrumented system expands, existing Receptors expand dynamically to handle new processes, and more Receptor instances can easily be added. JEDSI supports distributed collection and inspection of data from instrumented systems. Multiple Test Control processes and other clients of the Collector can simultaneously address separate visualization scenarios. Even the Collector components can be distributed using standard Java distribution mechanisms. In



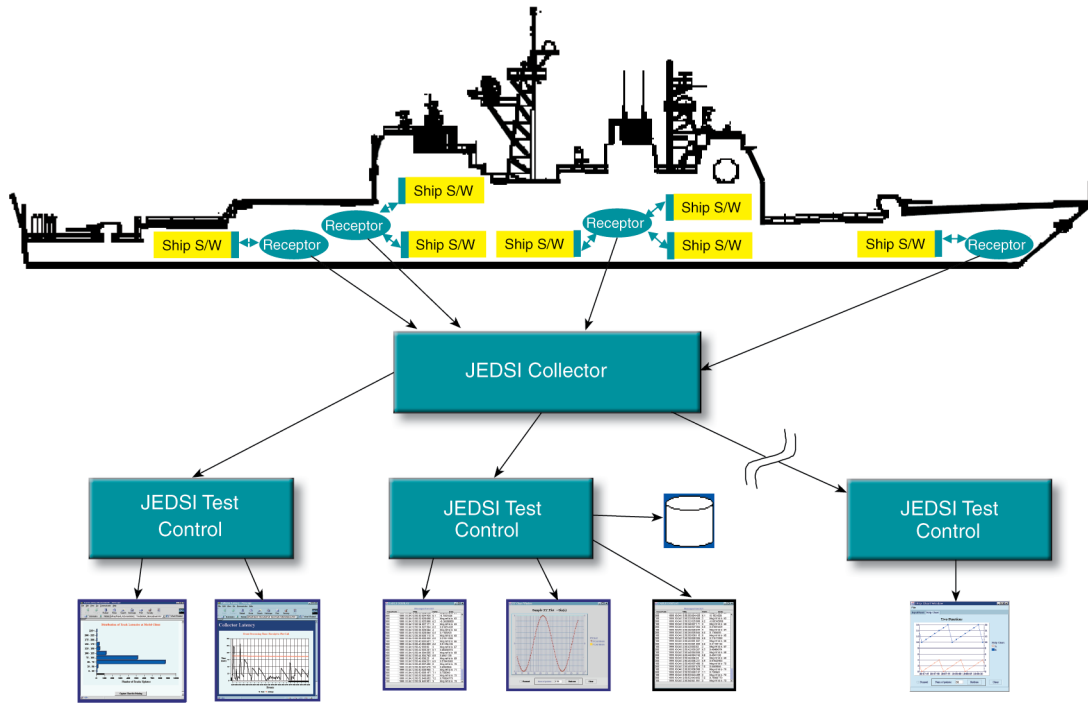


Figure 10. Potential future use of JEDSI in monitoring ship software (S/W) systems.

particular, the Evaluator objects that perform potentially computationally intensive processing can be distributed to other platforms to reduce the load on the Collector host.

JEDSI is a flexible tool. It is easily extended to specialized analysis by writing custom Evaluator tools that are automatically loaded at run time without any change to the core JEDSI system. It is dynamically adaptable at run time in response to user insights into the observed behavior of the instrumented system. It

provides simple, flexible displays that require only the knowledge of the data that are available for capture at the user-defined test points within the system under test.

REFERENCES

<sup>1</sup>Lange, F., Kröger, R., and Gergeleit, M., "JEWEL: Design and Implementation of a Distributed Measurement System," *IEEE Trans. Parallel Distrib. Sys.* 3(6), 657–671 (Nov 1992).  
<sup>2</sup>Shapter, B. A., *JEDSI Users Guide*, ADS-00-002, JHU/APL, Laurel, MD (Apr 2000).

THE AUTHORS



BARBARA A. SHAPTER is a member of APL's Senior Professional Staff. She received her B.S. degree from George Mason University and M.A. degrees from the University of Maryland, College Park (in mathematics) and The Johns Hopkins University (in computer science). As a software engineer, she has designed and developed applications from hospital clinical information systems at Johns Hopkins Hospital to e-commerce procurement systems for NASA and oceanographic model simulations at APL. Ms. Shapter joined the Advanced Systems Development Group at APL in 1997, where she has led the development of JEDSI. Her e-mail address is [barbara.shapter@jhuapl.edu](mailto:barbara.shapter@jhuapl.edu).



DOUGLAS G. CROWE is a section supervisor and a member of APL's Senior Professional Staff. He has B.S. degrees in computer science and electrical engineering, both from Pennsylvania State University, and an M.A.S. degree in administrative science from The Johns Hopkins University. Mr. Crowe has led groups in the development of automatic test equipment for airborne avionics, training simulators, and air defense command, control, computers, and intelligence (C<sup>3</sup>I) systems. He joined APL in 1996 and is currently the lead for the Laboratory's HiPer-D efforts. His e-mail address is [douglas.crowe@jhuapl.edu](mailto:douglas.crowe@jhuapl.edu).