Prototype Analyzer Architecture

The HP 16505A architecture allows multiple concurrent views of acquired logic analysis data. Markers on all views are correlated. The user only needs to place the marker on one view and the markers on the other views automatically relocate. Thus a stack anomaly in one view can be immediately correlated with the software routine causing the violation.

by Jeffrey E. Roeca

The HP 16505A prototype analyzer is the next-generation user interface for logic analysis. It is an X11/Motif application running on an HP 9000 Model 712 PA-RISC workstation directly tied to the HP 16500 logic analysis mainframe. This system is a managed, or closed, system. From boot until power-down this system is solely dedicated to extending the prototype debug paradigm from the nine-inch touchscreen of the HP 16500 to a multiple-window, high-resolution interface.

The design uses the X11/Motif graphical user interface (GUI). This was done to accomplish several goals. First, using an existing state-of-the-art GUI allowed us to focus on our own contributions. Second, with the distributed X11 interface, we can make the application available over a network to any X-compliant computer, including PCs. The software was written in C++ for the Model 712 workstation platform. This gave our application affordable MIPS, which were needed because debugging one-megasample logic analyzer traces across hundreds of channels will tax any computer.

Architecture

Fig. 1 shows a simple HP 16505A setup. There are three tools on the workspace. Two tools are analyzer machines, and one tool is a display tool. This simple configuration reveals the essential functions. Tools can be placed into a data flow and run. The analyzers are tools that probe target hardware. They sample real-world data and the display tools independently view that data.

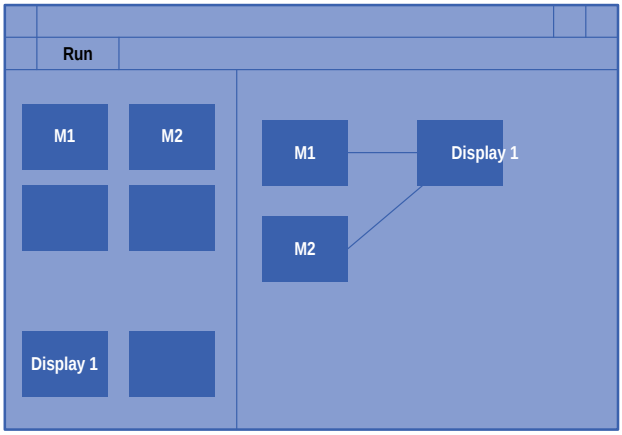

Fig. 1. *A simple HP 16505A prototype analyzer setup. Available tools are selected from the toolbox at left and dropped onto the workspace, placed into a data flow, and run. Here there are three tools on the workspace: two analyzer machines (M1, M2) and one display tool.*

To deliver this functionality we have partitioned the architecture into the following blocks:

- Tools. Tools are the atomic unit of value in the HP 16505A. All meaningful user interaction is performed within a tool definition.
- Acquisition Tools. The acquisition tool class is derived from the tool class. As such, it plays in the data flow workspace. The reason for making this a derived class lies in the way we connect or relate acquisition tools to the HP 16500 mainframe.
- Data. All tools export and digest a common unified data format.
- Workspace and Graph. The workspace is the visual programming GUI. It allows tools, represented by icons, to be connected to a data flow, represented by lines. The graph is the ordered list of tools in the flow.

- Run Management. This is a state machine that pumps data through the data flow by executing the graph.
- General Support. This is basically control or “glue” logic.

Data Flow

Let’s look at an example data flow through the architecture.

When the user powers up the HP 16505A we begin an HP-UX* operating system boot procedure. Most of the boot code is provided by the Model 712 ROM, but we adjust the daemons and hardware configuration for this application. Once the operating system is minimally configured we call the session manager routine, which never returns. The session manager handles software updates and spawning of the main application. This is also where the workstation is powered down. (The Model 712 workstation has a front-panel power switch, which forces the HP-UX file system to be written to the SCSI drive before actually cutting power. This prevents the user from crashing the file system. Unplugging the workstation can be problematic for the file system, however, so the session manager has a power-down button to remind the user not to simply unplug the product.)

The real application begins as main.c on HP-UX. We begin by creating an application context under the X Window System with a call to XtVaAppInitialize(). The entire application runs as an X11/Motif application. From the GUI mouse/keyboard callbacks through the acquisition control of the HP 16500, all software is run as an X11 application callback.

The HP 16505A system software offers a handful of services to tools. These services are collectively gathered beneath a structure called the frame. The frame is one of the few globally available pointers. Tools can access these system services by referencing frame->service. The frame services are:

- ResourceMgr. Colors, fonts, cursors, path names.
- RunMgr. State machine for stepping the tools through runs.
- FileMgr. Storing and loading of configurations.
- HelpMgr. Help screens.
- GeometryMgr. Pixel drawing management for the workspace.
- Symbols. Definition and display.
- FrameMarkerMgr. Global marker synchronization.
- FrameMessageExchange. Mail service between nodes on the graph.
- TbcMgr. Manager of toolbox icons.
- InstrumentMgr. Manages a list of instruments.
- AdminMgr. Networking and other HP-UX management.
- PrintMgr. Print services.

Next we perform a directory search for instruments and tools. All tools are compiled into shared libraries. These libraries exist beneath a well-known directory. We traverse the directories beneath this, searching for valid tools and instruments. A valid directory consists of a shared library, a resource file, and a pixmap. The load algorithm is generic and does not need to be rewritten when a new tool is created. Simply creating a directory with these three files is sufficient for this algorithm to attempt to load. Tools are noted, but are loaded only when placed onto the graph. This can be observed as a slight time delay in the creation of the first tool of a given class. The benefit of demand loading tools is that the memory is kept free for deep trace analysis.

Unlike tools, instruments are loaded when they are found. An instrument is an HP 16500. Within the HP 16500 there can be up to ten modules, but there are not usually so many and it is efficient and simple to load them at power-up.

Next the workspace is created. All of the loaded and noted tools are represented by icons in the toolbox.

Next we start the file system, remote procedure call mechanism, messaging services, and some other details. Finally, we submit all processing to the X Window System by calling XtAppMainLoop(). The application goes dormant until either a remote command or a GUI event activates a software callback. Even the remote interface becomes an X event as we receive remote procedure calls and simply pass them along to the X Window System. When XtAppMainLoop returns, the HP 16505A returns to the session manager.

Tool Design

Tools and their design are a major factor giving this platform room to grow. A tool is a C++ class with a small set of pure virtual functions. These functions are:

- getRev. What release of software (e.g., 1.20)?
- about. Returns general tool information in ASCII format.
- save. Saves the tools configuration to a file.
- load. Loads the configuration from a file.
- addToPopup. Call from the workspace frame to add to a menu.

- setName. Call from the workspace to change this tool's name.
- raiseAll. Open up or raise the windows.
- preExecute. Validates the internal state as legal to execute.
- validateConfig. Validates the configuration before loading.
- execute. Receives a new data group, returns a data group.

These tool functions are mostly mundane calls to get, set, save, or load static information within the tool definition. All real tools must supply these functions, and the system glue is written with calls through these functions from tool pointers kept in the graph.

Only one function deals with the primary objective of a tool—passing data—and that is the execute function. This function is declared as follows:

```
virtual DataGroup * execute (const DataGroup&) = 0;
```

A tool receives a call and is passed a pointer to the data and labels. The tool can then interpret, filter, or create data as needed. All that is required is that the tool return a pointer to the output data group. This is very simple and powerful. Compliance with the execution protocol does not require any coupling between tools.

From this definition all real tools are derived. Fig. 2 shows the current tool classes in the HP 16505A.

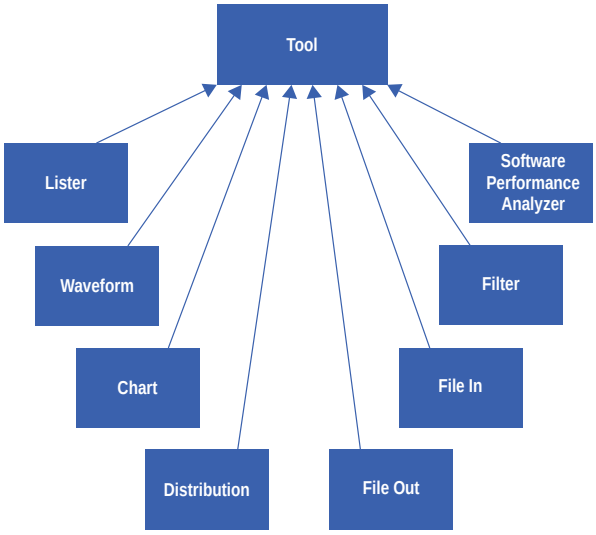

Fig. 2. *Tool class hierarchy.*

Acquisition Tool Design

The workspace manages an ordered list of tools. Consequently, a logic analyzer must be represented as a tool to be placed on the workspace. According to HP 16500 legacy, the fundamental working unit of a logic analyzer is a *machine*. A single HP 16500 card, or module, can have two machines. In the HP 16505A, each of the module machines is represented as an individual tool in the toolbox. The abstract C++ class for these machines is called a probed source. Probed sources must address multiple complications. One is communication with the HP 16500 module. Another is handling a workspace presentation that represents probed sources as independent tools, while behind the scenes both machines of a single HP 16500 module are joined at the card. Another complication is coordinating the remote acquisition of data. Fig. 3 presents the acquisition tool hierarchy, showing the basic composition of probed sources and their relationship to their module (card) and the HP 16500 (enterprise).

The class that is placed onto the workspace is one of the machine classes. Because these classes are derived from the probed source class, which is derived from the tool class, they can be inserted into the workspace. As previously stated, a logic analyzer card can have multiple analyzers, or machines. Accordingly, each machine has a relationship with its parent card class, which is derived from the abstract card class. Cards in turn are owned by the acquisition enterprise. The enterprise class is derived from the abstract instrument class. Within the instrument class exists the transport knowledge, which happens to be SCSI.

Encapsulating the SCSI transport in the instrument class allows us to port the transport layer with a minimal impact on the rest of the software. This was a benefit in the initial stages of development when the SCSI interface did not exist and we programmed the HP 16500 over the HP-IB (IEEE 488, IEC 625).

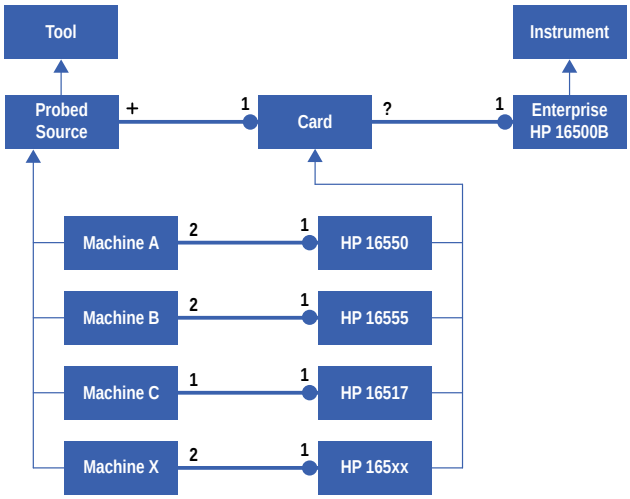

Fig. 3. *Acquisition tool class hierarchy. The dots and numbers are Booch notation for N-to-1 relationships. For example, an HP 16550 card has two machines.*

By using the existing ASCII programming strings of the HP 16500 we are able to acquire data in an existing stable platform. By using the SCSI transport we minimize the latency of transfer from the HP 16500 to the HP 16505A.

Data

Data is different for each analyzer card in the HP 16500. Historically this slowed development by giving rise to custom, hardware-specific algorithms to render that data. Behind similar HP 165xx interfaces may be completely different software. Fig. 4 shows the transform that we apply in the HP 16505A prototype analyzer. As data is uploaded from the unique hardware, it is transformed into a common, or normalized format. This normalized data is what all tools downstream from the analyzer source operate upon (see ***Article 4***).

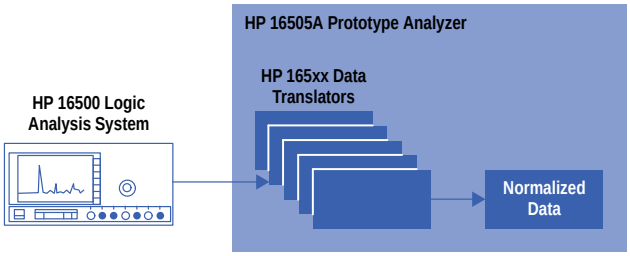

Fig. 4. *Data is different for each analyzer card in the HP 16500 logic analysis system. As data is uploaded into the HP 16505A prototype analyzer from the unique hardware, it is transformed into a common, or normalized format.*

Workspace and Graph

The primary role of the workspace is to construct the data flow relationship that is executed during runs. The data flow is constructed with tools and data. The GUI provides a toolbox that contains icons representing all available tools. A user drags an icon representing an available tool from the toolbox and drops it into the workspace. Data flow is represented by lines connecting the icons or tools. Each tool that is placed onto the workspace is entered into the graph. The graph is an ordered list of containers. A container is a structure that holds information about real instantiated tools. Any software function can execute the graph, which results in an ordered delivery of containers to the requesting function. The function can then call the appropriate tool function through the container.

The graph has a simple set of rules determining the order of execution of the list of tools:

- Only tools that are on the workspace are executed.
- Orphan tools—tools that have no data connections—are executed first.
- Top-level tools such as logic analyzers and oscilloscopes are executed next.

- The graph does not transition from one level to the next until all tools at a given level are complete. This ensures that any tool that receives data from multiple sources will have the data available when the tool is executed, since all parent tools must have completed before the tool is called.

Fig. 5 shows a workspace representation with three analyzer machines, two filter tools, a lister, and a waveform tool. M1 and M2 are merging their independently acquired data together into a single data group and then passing that merged data group through the Filter1 tool to the lister. M3 is an independent data group that passes through Filter2 into the waveform tool. There are highlighted boxes labeled level 1, level 2, and level 3. These boxes indicate the order of execution of the graph. Level 1 tools are the data sources, or analyzer tools. This level is executed first. No tool in level 2 or level 3 will execute until all level 1 tools are complete. Once level 1 is complete then the level 2 tools will execute. No tools in level 3 will execute until level 2 is finished. This is a general rule: level N–1 must finish before level N executes.

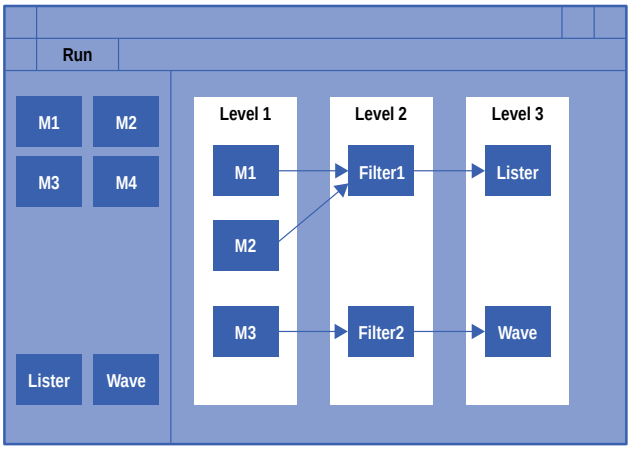

Fig. 5. *A workspace representation with three analyzer machines, two filter tools, a lister, and a waveform tool.*

Run Management

The run manager is the software that pumps data through the graph. It is a state machine that executes the graph. Remember that executing the graph is an ordered, level-by-level event. This ordering allows for all probed sources to be configured, started, and uploaded as a coordinated group.

Fig. 6 shows the states and their execution cycle, together with the entry points into the state machine. The states are:

- PreRunDownload. The HP 16505A probed source software maintains internal configurations. The GUI modifies the configurations and the changes are stored in the HP 16505A without remotely updating the HP 16500. When Run is pressed, a probed source is requested to download to the HP 16500 the current HP 16505A probed source configuration.

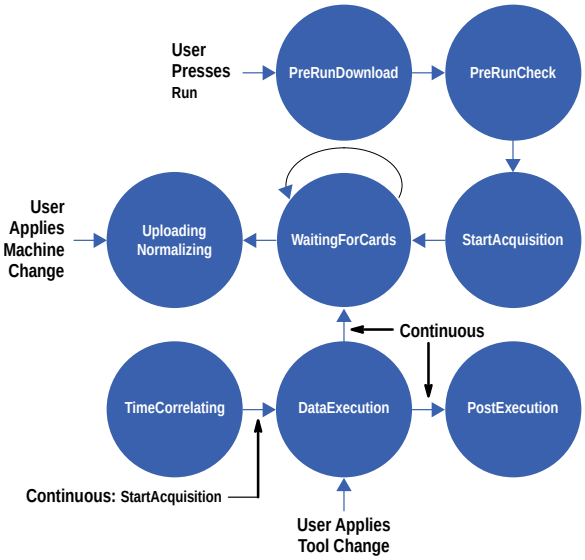

Fig. 6. *Run manager state diagram.*

- PreRunCheck. A given HP 16500 card may have an illegal setting that prevents a run from proceeding. This state allows the HP 16500 status to abort a run.
- StartAcquisition. The HP 16500 is started. The current release of the HP 16505A only performs group runs. A group run is one that time-correlates the trigger events from participating modules.
- WaitingForCards. The HP 16505A polls the HP 16500 run status waiting for the run completion.
- UploadingNormalizing. Each card is called serially to upload and normalize the acquisition data.
- TimeCorrelating. After each card is normalized the system acquires the timing correlation values and updates the normalized data.
- DataExecution. The normalized data is now passed through the graph to the configured tools. Each tool called will render the data and update its display.
- PostExecution. The HP 16505A does some cleanup.

The graph is executed during each state at least once. A given state may only choose to communicate with a tool if the tool is a probed source. A state may execute the graph multiple times. For example, WaitingForCards is a state that can take a significant amount of time, depending upon the repeatability and cycle time of the trigger specification. The run manager could be stuck in this state forever. The logic within this state executes the graph. If the cards are not complete then the run manager sets an X timeout and exits.

The setting of the X timeout is significant. It is essential that the GUI is allowed to run to service potential stop events. At the first release there is no remote control of the HP 16505A, and a GUI event on a stop button is the only way to abort a run.

Example Control Flow through the Software

Fig. 7 shows a simplified control flow for acquiring status during an acquisition of data. A user has previously pressed the Run button, which starts the run manager state machine. In this example the run manager has been called (from an X timeout) and is processing the WaitingForCards state. This state, like all run manager states, calls the graph execute function. This causes the graph to cycle through all of the tools on the workspace. The run manager calls tools that are probed sources requesting their status. Now is where the mapping of analyzer machines, or probed sources, to cards becomes important. The HP 16500 remote control is ordered by modules. Within a module, communication is with an analyzer machine. The HP 16505A has requested the run status of an executing machine. This request is passed to the parent card class. The card knows which HP 16500 slot to select. The card also knows what messages to apply to obtain the status. These ASCII HP 16500 command strings are then passed to the enterprise, or HP 16500B class. This is the class that inherits from the abstract instrument class. The SCSI transport sends the ASCII programming instructions down to the HP 16500B and there they are parsed and interpreted.

Fig. 8 shows a similar simplified block diagram with the normalized data, tools that are not probed sources, and the tool support blocks added. Ignoring the fine details of the myriad of support tools, correlated markers, and so on, one can obtain a good understanding of the HP 16505A architecture from this figure. The frame object has a run manager object which controls the state machine for acquiring and pumping data through the tools. The workspace has an ordered list of instantiated tools and controls the execution order. Tools are either independent tools or probed sources. If they are probed sources they communicate through their parent card through their instrument through the transport layer. Probed sources generate normalized data, and all other tools can use, modify, and generate normalized data.

Examples of Visualization

The primary value of the HP 16505A lies in its ability to help the user visualize data. The architecture was defined to allow multiple concurrent views of the acquired data. By slicing and dicing data in different ways, fault isolation becomes easier. Fig. 9 shows an HP 16505A setup demonstrating how visualization in multiple views promotes rapid time to insight. Software stack corruption is a difficult problem to solve. Usually a corrupt stack does not directly reveal itself. More often, the product crashes far away from the actual write violation. In this example a 68332 microprocessor running software has been sampled. This data has been displayed both unfiltered and filtered. The filtered data is shown in ASCII format, as a chart, and as a distribution. The filtered data in chart mode shows the stack as a function of time. The filter is a simple range between the low and high boundaries of the stack. Visually one can see an anomaly in the stack space. In this example the write taking place beneath the G1 marker in the Stk vs. Time window is in error. By looking at the ASCII listing of the unfiltered data in the Raw Trace window we can see exactly what software was executing at the time that this erroneous write to the stack space took place.

The HP 16505A allows these multiple views—the lister and the filtered chart—to be displayed concurrently. Within the HP 16505 display tools we have correlated markers. The default behavior of a display tool is to track the placement of a marker automatically. The user only needs to place the marker on the visual stack anomaly, and the lister automatically relocates itself to reveal the software routine in violation.

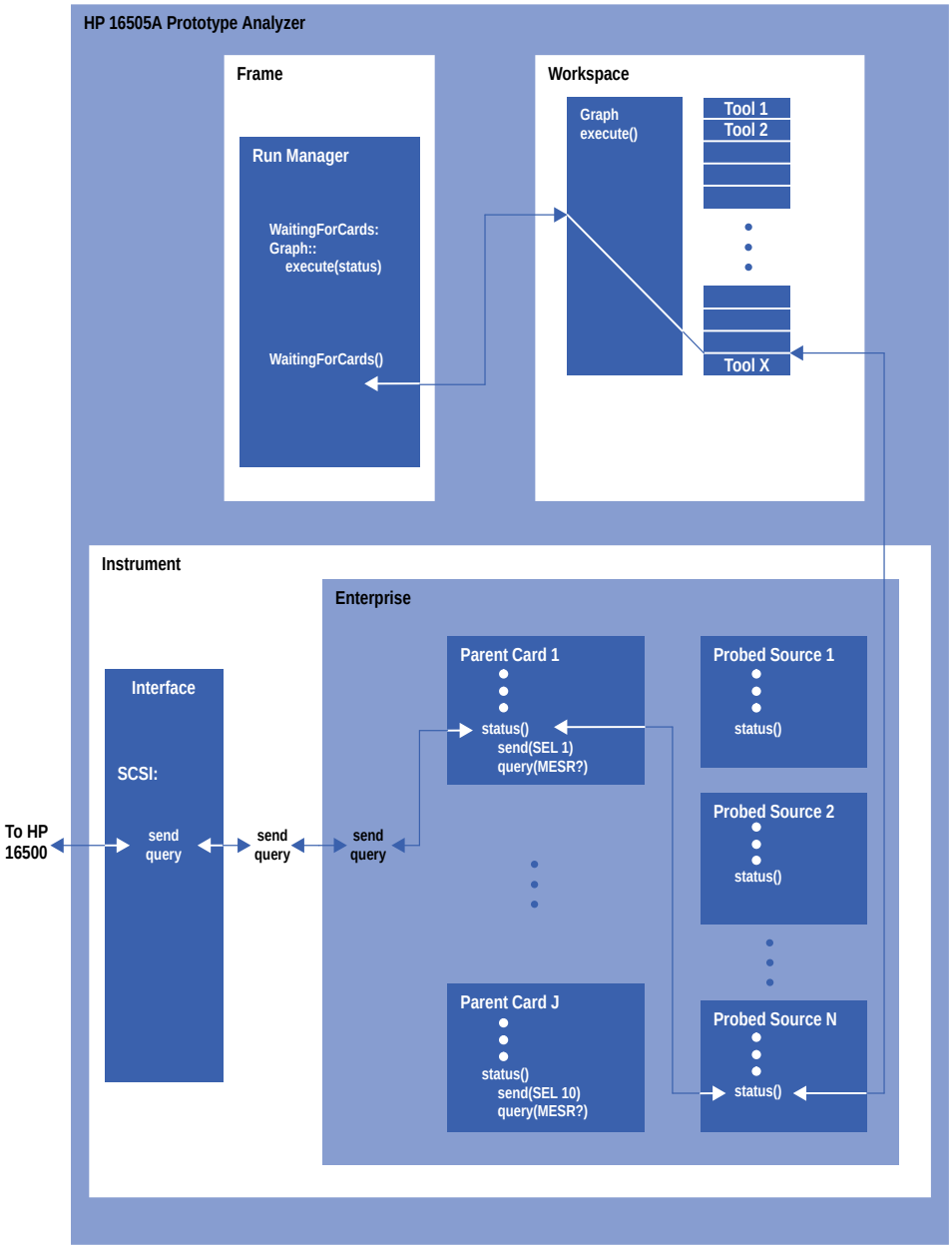

Fig. 7. *A simplified control flow for acquiring status during an acquisition of data. The run manager state machine has been called and is processing the* WaitingForCards *state. This calls the graph execute function, causing the graph to cycle through all of the tools on the workspace.*

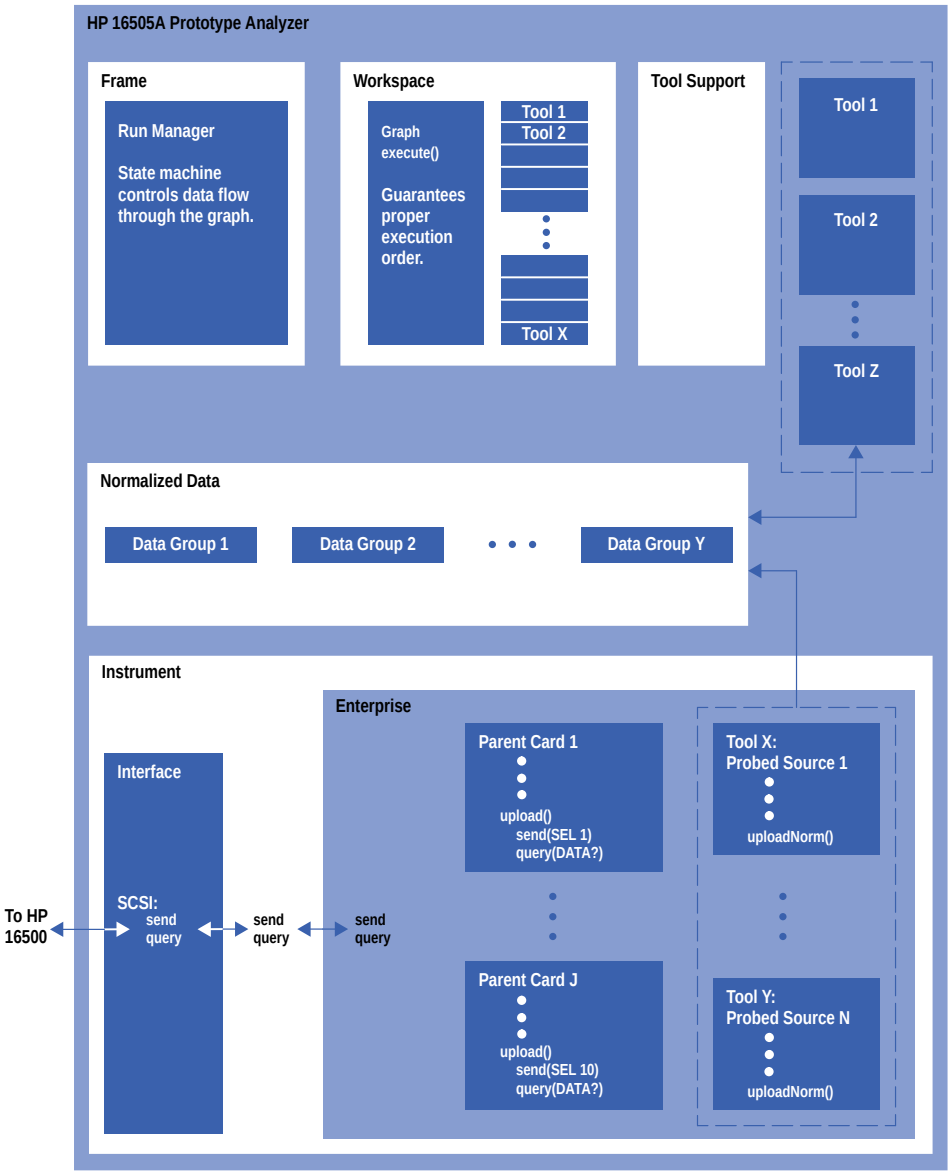

Fig. 8. *Simplified block diagram similar to Fig. 7 with normalized data, tools that are not probed sources, and tool support blocks added.*

Fig. 9. *An HP 16505A setup demonstrating the diagnosis of a software stack corruption problem. The write taking place beneath the G1 marker in the Stk vs. Time window is in error. By looking at the ASCII listing of the unfiltered data in the Raw Trace window we can see exactly what software was executing at the time that this erroneous write to the stack space took place. The user only needs to place the marker on the visual stack anomaly, and the lister automatically relocates itself to reveal the software routine in violation.*

Acknowledgments

The author would like to acknowledge James Kahkoska for defining, designing, and driving the product idea into reality. I would also like to thank Mark Schnaible for helping distill the HP 16505A architecture down to a few slides.

▶ Go To Next Article

▶ Go Table of Contents

▶ Go To HP Journal Home Page