

As published in (and reprinted with permission from)

**INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED
MANUFACTURING, 2000, VOL. 13, NO. 3, 225-244**

<http://www.tandf.co.uk/journals>

The National Electronics Manufacturing Initiative (NEMI) plug and play factory project

A. Dugenske, A. Fraser, T. Nguyen and R. Voitus

Abstract. The National Electronics Manufacturing Initiative's (NEMI) plug and play Factory Project addressed the issues of how to quickly integrate new pieces of electronics assembly equipment into a shop floor line management system and how to manage the vast amounts of data available in today's electronics manufacturing environment. The necessary technical infrastructure was designed, developed, and demonstrated over a two-year period. New standards activities for electronics manufacturing were initiated where existing standards were either non-existent or insufficient to achieve the project goals.

1. Background

The National Electronics Manufacturing Initiative is a US-based consortium of 55 electronics companies. The goal of the consortium is to increase the competitiveness of US electronics manufacturers. NEMI develops five-year technology roadmaps and performs gap analysis to identify critical needs of their original equipment manufacturer (OEM) and electronics manufacturing service (EMS) members. New projects are initiated based on the results of the gap analysis. The plug and play Factory project was started under the auspices of the Factory Information Systems Technical Implementation Group in December 1997. The goal of the project was to design, develop and demonstrate a technical infrastructure to enable the replacement of both electronics assembly equipment and electronics assembly software in a fraction of the time and at a fraction of the cost normally associated with those activities.

2. Project Goals

The goals of the project were to reduce the amount of time and cost that it takes to integrate a new piece of electronics assembly equipment into a shop floor environment and start collecting data and controlling that equipment. It is estimated today that the integration cost of a typical factory information system is up to four times the cost of purchasing that system in the first place. Also, the possible two years that it typically takes to design and implement a new factory floor system is much longer than the product technology life cycles of today's electronic products. Often what happens after a long period of analysis, design and implementation is that users declare "It is just what I asked for, but not what I want". The reason for this is that the business model has changed so drastically during the time it took to implement the factory information system that the system is at best underutilized and at worst never used to support real production.

By demonstrating an order of magnitude reduction in the amount of time and the cost of implementing a new factory information system on an actual electronics manufacturing line, the project achieved its goal of drastically reducing the break-even point for implementing a new factory information system.

3. Project Structure

The project was structured as a working group of 16 NEMI member companies and organizations that each contributed both time and money to the project. Gaining the necessary resource commitment as part of the entry fee for the project was critical to its success, as the expertise and insight that was provided by the participating companies proved invaluable at several critical points during the project. The following companies all participated in the project: AMP, Celestica, Compaq, Delphi Delco, EDS, GenRad Inc., Georgia Institute of Technology, Industrial Computer Corporation (ICC/GR Software), Intel, Lucent Technologies, National Institute of Standards and Technology, Sandia National Labs, Solectron, Speedline, State University of New York, Binghamton, and Universal Instruments Corporation.

Project meetings were held every two months and rotated through the different project member sites. This strategy also proved very beneficial, as the project members were able to gain wider visibility for the project in their respective companies while at the same time comparing their own experiences to those of their peers at the other project member companies.

Key to the project also was the element of a testbed on which to test the ideas developed by the other project teams. There were three project teams in all. The first team was the Equipment Communication team which was led by Delphi Delco and Lucent. This team developed the performance and cost requirements for the project. The second team was the Framework Development team. This team was led by GenRad, Inc. and Universal Instruments Corporation and was responsible for developing the technical infrastructure used on the project. The third team was the Testbed team. This team was led by the Georgia Institute of Technology and was responsible for testing the concepts developed by the Framework Development team and determining if met the needs as specified by the Equipment Communications team. This feedback mechanism between the requirements, design, and implementation teams served as a rapid prototyping environment and helped the project focus its resources and kept it moving forward by quickly either proving or disproving the technical feasibility of several different technical approaches investigated during the project. The interaction between the Framework Definition task and the Testbed task is shown in figure 1.

By making use of a testbed, it is felt that the standards development process can be greatly accelerated. Early adopters of the standard can also approach the standard with lower risk and at a lower cost, since many of the initial inconsistencies, incompatibilities and over-sights can be found prior to the recommendations being released as a standard. Companies that make use of the standard will have the confidence to adopt that standard knowing that the fundamental technologies have been tested prior to their implementation. The testbed can also assist organizations to adopt the standard by providing an initial reference implementation. The cost to individual companies is also lower, since many participants share the initial rather than develop something independently that may or may not become widely adopted at a later date. This is especially true of many small to medium size companies that provide much of the software used in electronics manufacturing, since their budgets are less likely to absorb repeated changes in technology direction.

This technique is different from conventional standards development efforts in the sense that recommendations are tested and fed back to the recommending body in a very short period of time (months). When standards are developed without the use of a testbed or reference

implementation, the recommending body doesn't receive feedback until an organization puts the standards into practice, a process that can often take years.

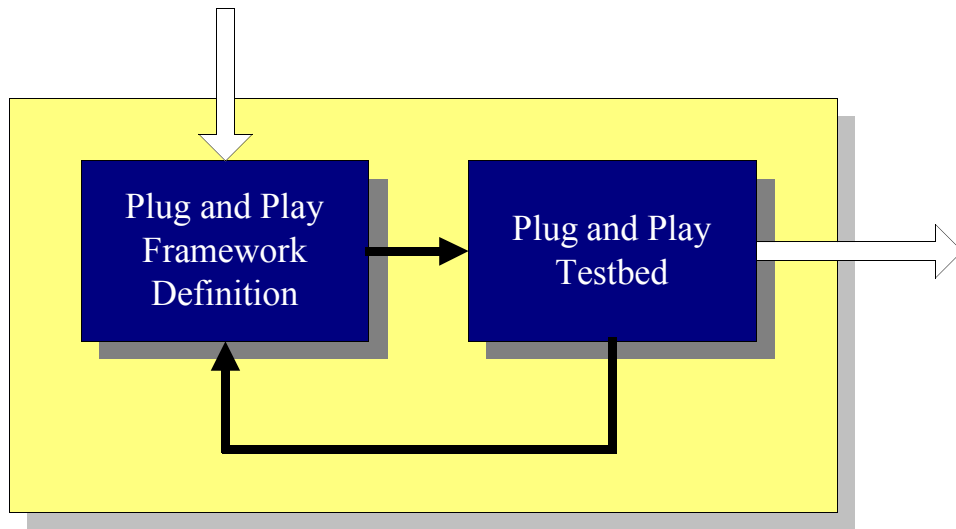


Figure 1. Plug and play project task interactions.

4. NEMI plug and play factory project requirements

4.1 *Motivation*

More efficient SMT assembly operations, with increased equipment utilization and reduced quality problems, are the primary reasons an OEM or EMS would consider implementation of a line-wide monitoring application. While vendor-specific monitoring packages have been available, such packages do not address the desire to install the best-suited machines on a SMT line, regardless of the vendor. What is needed is a "plug and play" environment, where equipment from multiple vendors can be installed and a generic interface used to connect the equipment to a vendor-independent line monitoring host application.

Such a scenario has been successfully implemented in the semiconductor industry. Process monitoring, equipment status and parameter tracking are critical to the rapid achievement of profitable wafer yields. The wafer fabrication process, as much an art as a science, demands close monitoring of each operational step. The Generic Model for Communications and Control of SEMI Equipment (GEM) interfacing standard enjoys broad vendor support in the semiconductor industry and allows implementation of plug and play integration of multiple-vendor equipment when configuring a production line. GEM builds on the Semiconductor Equipment and Materials International (SEMI) organization SEMI Equipment Communications Standard 2 (SECS-II) by defining the messages and behaviors of semiconductor manufacturing equipment. Many surface mount equipment vendors have attempted to adopt the GEM/SECS-II standards for their offerings but the efforts have not resulted in a true plug and play environment. The NEMI plug and play factory project is an attempt to overcome the difficulties and achieve similar benefits in the printed circuit board assembly area of the electronics industry.

4.2 *Problems with GEM*

GEM implementations on printed circuit board assembly lines are rare. As described in the last section, OEMs are motivated to implement increased monitoring of their printed circuit board assembly lines. So why are GEM implementations so much less pervasive in the printed circuit board assembly arena than in the semiconductor industry? Reasons include costly interfaces, poor implementation of GEM standards, and lack of off-the-shelf host packages, leading to a weak cost-benefit position. This section will examine these issues.

4.3 *SMT GEM interface costs, availability and support*

The GEM option for a single piece of SMT equipment can range from \$1,000 to \$10,000. This is for a TCP/IP interface card and the GEM interfacing software. Run-time component licenses alone average almost \$1,000. Contrast this expense with the costs associated with a network interface for a PC, typically under \$200, including interface card and software. Customer demands and production volumes are responsible for the discrepancy. The market for SMT equipment GEM interfaces is in the tens of thousands while the PC interface market is in the tens of millions. Furthermore, the benefits of providing networking capabilities on a PC are well known, particularly with the emergence of the World Wide Web. These benefits are available immediately upon installation of the PC networking interface. With the SMT equipment GEM interface, the lack of off-the-shelf host software means the installation of the interface is only the first step in a lengthy and expensive process to collect all the data necessary to monitor the entire surface mount process.

Acquiring robust GEM interfaces for SMT equipment remains problematic. Interfaces are generally available for only the newest equipment models. Even when GEM interfaces are available, they are not well supported by the equipment vendors. More than one equipment vendor provides a line monitoring host software package that makes use of proprietary interfaces to the vendor's machines, even when the vendor also offers their own GEM interface as an option.

4.4 *Interpretation of SEMI standards*

GEM documentation for SMT equipment typically includes information on state models, implementation of messaging, and equipment data elements. State models are used to describe the behavior of the equipment. Typically, individual state models are provided for communications, control, and processing characteristics of the equipment. Standards implementation issues exist for all three models. Examine the state models in GEM manuals for any two similar pieces of SMT assembly equipment from different vendors and you will quickly see the degree of interpretation that has been applied to various sections of the SEMI GEM standard. Consider the ten states/sub-states depicted in the Communications State Diagram from the SEMI E30-95 standard as shown in figure 2. This state model describes how communications between the equipment and a host application can be established.

While a detailed tutorial on the Harel state diagram notation adopted by SEMI is beyond the scope of this paper, the following basic description will provide sufficient detail to understand the points being made. Those desiring a more complete understanding should see Harel (1987).

The state model diagram represents the possible states of a system and the transitions from one state to another. States are represented by rectangles. Sub-states are represented by a rectangle wholly contained within a larger rectangle, the parent state. Sub-states are mutually exclusive, i.e. the system can only be in one sub-state at a given time. An exception, where a system can be in two sub-states simultaneously, is indicated by sub-states separated by a dotted

line. Events, represented by arrows, trigger the transition from one state to another. The numbers on the arrows correspond to entries in a companion State Transition Table. Finally, an arrow emanating from a solid circle is a default entry point and an arrow terminating in an open circle containing a C indicates the sub-state that will be entered is conditional on a configurable parameter of the system.

Figure 2 indicates that a piece of equipment has two communications sub-states, Enabled and Disabled, and that the one entered upon power-up depends on a configurable parameter. The state diagram and associated state transition table (not shown) fully define the activities and responses associated with both host-initiated and equipment-related attempts to establish communications.

Most vendors simplify the Communications State Diagram to just four states-sub-states as shown in figure 3. But critical details associated with establishing the communications are thus left undefined. The developer of a host application will have to determine the unspecified communication state details by trial and error, and will often have to generate unique software code to handle each different piece of equipment.

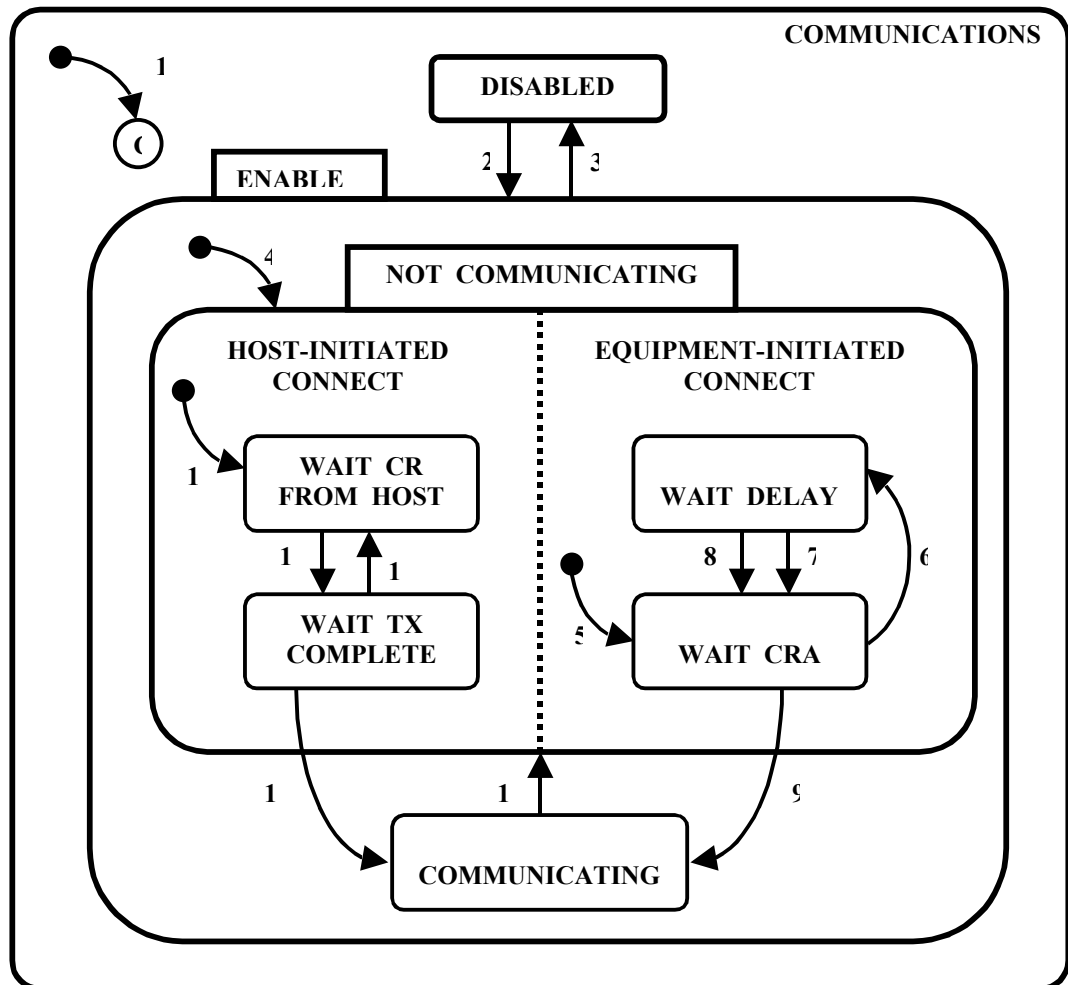


Figure 2. SEMI standards document SEMI E30-0998. Republished with permission of SEMI. SEMI standards are reviewed at least once every five years.

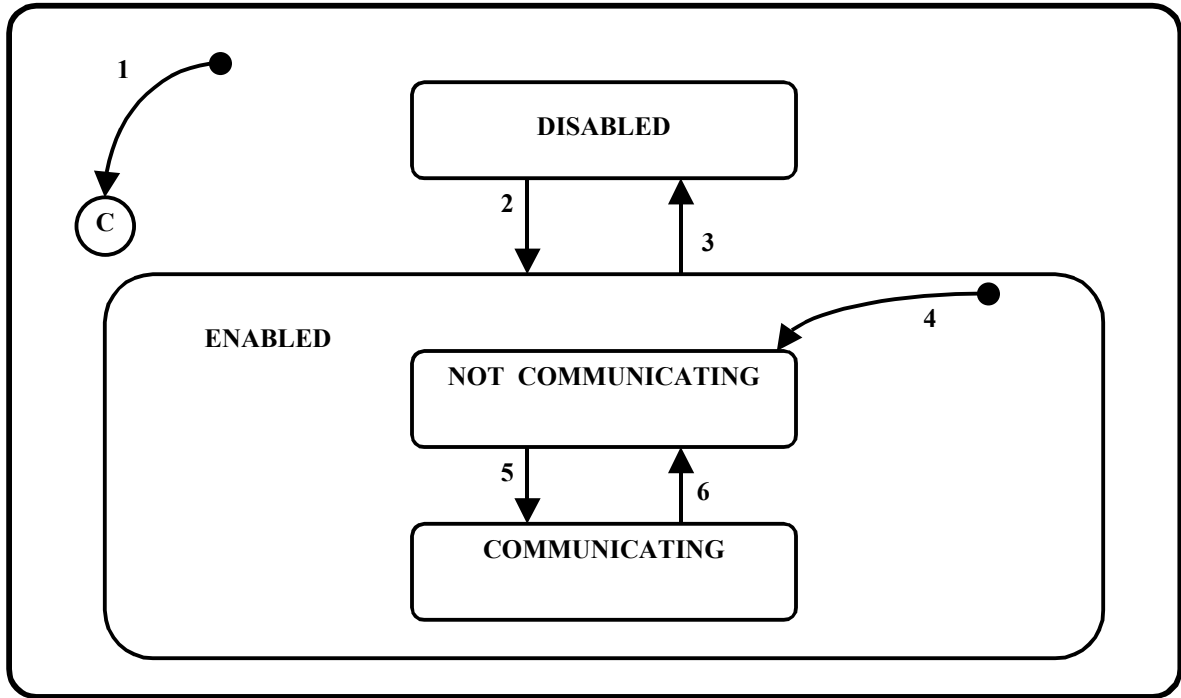


Figure 3 – Typical vendor GEM state diagram.

Table 1. Vendor variation.

	Vendor 1	Vendor 2	Vendor 3	Vendor 4	Vendor 5
Name	GemTime	Time	Time	GemTime	Clock
ID	17	39	31	30250	3
Size	12 bytes	12 bytes	12 bytes	12 bytes	16 bytes

Other vendors omit the Communications State Diagram entirely, merely including a statement that their interface complies with the GEM specifications. Unfortunately for the end user, the devil is in the details of the implementation. Regardless of what is portrayed in the equipment GEM manual, some implementations of the Communications State Model prevent booting up the particular piece of SMT equipment in a state where communication is enabled, forcing button-pushes at the equipment operator console to initiate communications between the equipment and the host.

Moving beyond the simple Communications State Model, the situation only gets worse. Most equipment vendors lift the Control State Model directly from the SEMI standards documentation but some vendors feel compelled to renumber the transition arrows illustrated in the model. To add to this confusion, the vendor then refers the reader to the SEMI standard Control State Transition Table for details of transitions depicted in the diagram. The transition numbers in the diagram and the table do not correspond, leaving the implementers on their own to define states and necessitating the development of additional custom code.

Major variations appear when the vendors address the Processing State Diagram. Some of the variation can be justified by differences in the physical characteristics of the equipment being described. However, the processing state diagrams for equipment of similar complexity

can depict from 5 to 35 unique processing states/sub-states, depending on the level of detail provided by the vendor. The user is left with the mapping of these states to the desired set of common states that will enable monitoring of the status and utilization of the equipment on the line in a consistent, concise manner.

Message formatting is the area where there is the greatest uniformity among the various SMT equipment vendors' GEM implementations. The GEM standard, which specifies the various messaging Streams and Functions, is generally followed. Still, syntactical differences in the GEM manuals needlessly complicate forming a complete understanding of messaging implementation details from multiple vendors.

Finally, message content presents another area where implementation differences among vendors make integration of SMT equipment on a line difficult. A data item may be specified as an ASCII item of 20 bytes maximum by one vendor, 30 bytes max by another, and 40 bytes max by a third. The name, ID and size of a generic variable containing the date and time reported by the equipment vary from vendor to vendor as illustrated in table 1. Such arbitrary differences can be addressed by a line monitoring host configuration software but are not compatible with the notion of plug and play. Note: those desiring a complete understanding of the GEM/SECS state models are referred to the SMEI standard referred to in the caption of figure 2.

4.5 *Generic host applications*

Until recently, someone seeking to implement a line monitoring application involving equipment from multiple vendors was faced with two alternatives: in-house development or custom software from a systems integrator. A prototype basic host monitoring application for a typical SMT line can cost \$200,000 or more when both system integrator and internal resources are considered. Add the costs of expanding the application to additional pieces of equipment and/or additional lines, monitoring more than just basic parameters and documenting the administration and maintenance issues, and the total cost can exceed \$500,000.

Off-the-shelf line monitoring host applications supporting multiple vendor SMT lines have begun to appear and most use GEM where appropriate. Such applications fall into the \$50,000 - \$100,000 range per line. However, due to the difficulties outlined above, the non-GEM interfaces are used in many, if not the majority, of the equipment connections. This partially negates the plug and play advantages of the generic host interface. There is little prospect for improvement in this area unless equipment users demand functional GEM interfaces or alternative solutions are developed.

4.6 *GEM cost analysis*

Given the high costs, both in terms of interfacing expenses and host application effort and expense, the GEM cost-benefit position has not been as attractive as that in the semiconductor industry. Perceived benefits fall short as setup and processing errors in printed circuit board assembly tend to be more readily detectable, impact fewer units, and result in less scrapping expense than similar errors in wafer fabrication. This relatively weak cost-benefit situation, coupled with other selection criteria, often that means the buyers of new surface-mount (SMT) equipment will not push very hard for a quality GEM interface on the machine they are considering. Unfortunately, without user insistence on working plug and play capability, vendors see little incentive to provide working GEM interfaces. Many vendors offer a proprietary interface alternative, which locks the end users into a single equipment vendor.

As SMT features become smaller and outsourcing complicates program management, the potential for costly errors increases. Outsourcing of printed circuit board assembly production may actually increase OEM interest in line monitoring applications as visibility into the EMS operations is sought. Both trends will contribute to an increasing demand for plug and play capability on SMT equipment. However, the implementation and operational limitations that exist with the GEM interface will remain and improvements will be slow in coming. Alternatives to the GEM interface are being investigated and are described in the remainder of this paper.

5 *NEMI plug and play factory design*

5.1 *Overview*

The wide-spread adoption of the Internet and its related technologies has provided companies with a universal and open architecture that can be used to establish business-to-business links and a new distributed enterprise computing platform.

Applied to the NEMI plug and play architecture, electronics assembly equipment will no longer be considered only as a production tool, but also as a fully functional information asset. The technical advantages of this architecture (portability, standardized user interface behavior (browser), and low administration costs) will enable new shop floor line management and remote field service applications.

The NEMI plug and play framework was built around the idea that the Internet itself is a distributed computing platform. The technical infrastructure to allow both hardware and software to be replaced was designed, developed, and implemented. A depiction of the NEMI plug and play framework is shown in figure 4. The project also developed Equipment Interface Models as alternatives to GEM, so that the shop floor equipment can communicate directly with the NEMI plug and play framework. Both the Equipment Interface Models and the framework are based on the HyperText Transfer Protocol (HTTP) and the eXtensible Mark-up Language (XML). The initial investigation shows that communication interfaces that make use of HTTP, XML, and the web server technology are orders of magnitude simpler and less costly to integrate into a factory information system, while providing the key functionality of a GEM interface. The development of web technology has greatly simplified the development of client/server applications. By implementing pure web-based equipment communication using the HTTP and XML protocols, commercialization should be possible at a much lower price point and with a much wider available resource pool than is possible with GEM.

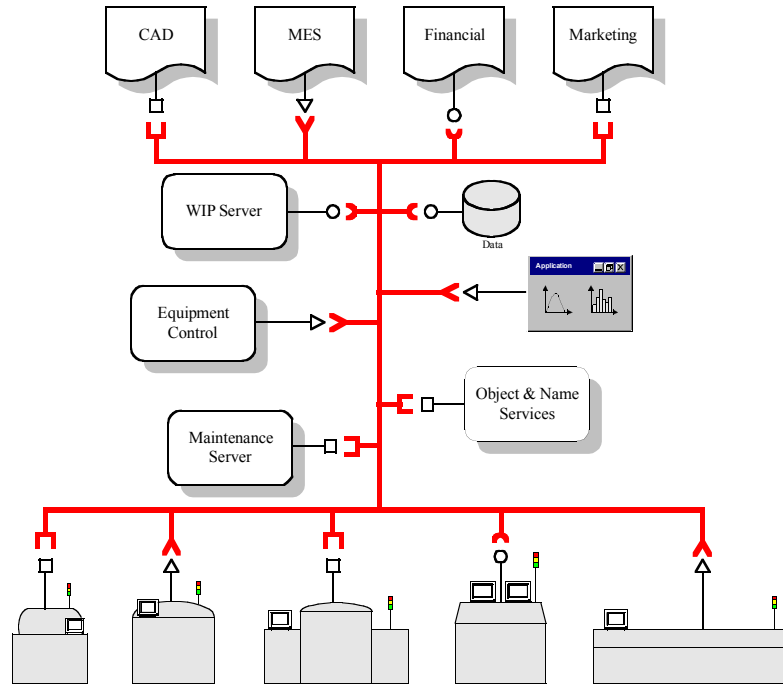


Figure 4 NEMI plug and play framework.

5.2 Implementation strategy

As state earlier, the proliferation of web technology has greatly simplified the development of client/server applications over the Internet. Recent developments have extended the protocols for use by electronic commerce applications. These same technologies promise to allow similar advantages for use in factory floor applications.

The key advantages of the Internet-enabled HTTP protocol and XML are the reduced cost and time of deploying systems dependent on machine communications. The plug and play project combined these advantages while leveraging the experience gained by working with the SEMI specifications, especially the GEM message structure and semantics.

As the Internet becomes more popular, net traffic will probably continue to increase dramatically, leading to unavoidable transfer delays and unpredictable reaction times. Due to the time delays inherent in operating on the Internet, this project focused only on the deployment aspects of a plug and play framework with an intranet or an extranet. The important issue of equipment and data security also means that practical implementations of the NEMI plug and play framework would take place on intranets or extranets where data and equipment are protected by firewalls and network response times can be controlled. It is also important to note that the NEMI plug and play framework's primary characteristic is information gathering, which requires information to be delivered with a time step of seconds. It is not a control framework (i.e. servomotor control) which would typically require a reaction time of the system of the order of 20 milliseconds.

The area of remote control of equipment has been also clearly declared out-of-scope for this project. Remote control of equipment over the Internet (especially start-up) raises severe safety issues.

5.3 Use of eXtensible Mark-up Language (XML)

While HTTP is already a proven standard communication protocol responsible for the rapid expansion of the World Wide Web, XML adds dynamism to static HTML documents and addresses some of the other deficiencies of HTML regarding conveying document structure and content.

Simply by writing an XML file, the user is structuring information and building a hierarchy of objects. Links are inherently flexible and can represent any data structure. XML enables the creation of web documents that preserve data structure and provide “machine-readable” information to facilitate web automation. Today XML can guarantee platform-independent inter-application data interchange. XML is becoming an enabling technology on the Internet, transforming it from a static medium to a powerful infrastructure for collaborative and interoperable applications.

XML enables the creation of web documents that preserve data structure and provides “machine-readable” information to facilitate web automation. XML has received widespread support as a standard. It is a self-describing data format that can accommodate complex data structures. The main downside of using XML as a message format is its verbosity. Widely accepted compression techniques and the benefits of standardization will outweigh this disadvantage.

XML describes a document by using a document type definition (DTD) or a document content description (DCD). In this paper we only discuss the DTD, which may be thought of in this context as a contract between two communicating systems such as a machine control and a host scheduling system. Before sending a message, a computer can check the syntax of the message for correctness based on the DTD. The receiving computer also uses the DTD to validate the message and parse it for useful information. The XML DTD provides an unambiguous description of the data format that can be easily parsed by receiving programs on different hardware platforms.

In the web-based equipment interface model development as an alternative to GEM, event messages coming from the equipment are converted in the XML format and embedded in XML messages, so that these events can be broadcast among framework participants.

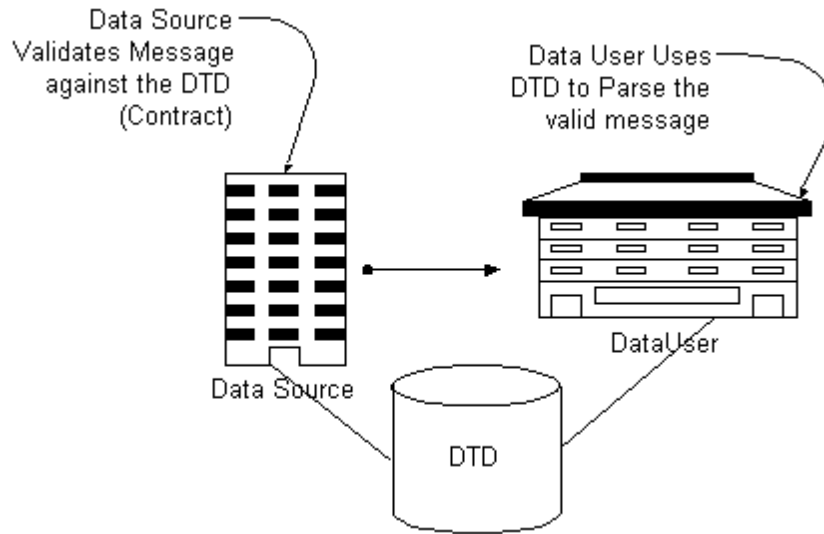


Figure 5 : XML document type definition.

NEMI Testbed DTD-file:

```

<!--NEMI Testbed DTD Version 1b-->
<!ELEMENT NemiMessage (Control, Data)>
<!ELEMENT Control (Sender, Receiver+, Command, Type, Status,
DateTime)>
<!ELEMENT Data (Charting | Equipment | Line | Materials |
Operation |
    Order | ProcessPlan | Product | Report | Resource |
    Task | WorkInstruction)>
<!ELEMENT Sender (#PCDATA)>
<!ELEMENT Receiver (#PCDATA)>
<!ELEMENT Command (Get | Set | Event | Response)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT DateTime (#PCDATA)>
<!ELEMENT Get (#PCDATA)>
<!ELEMENT Set (#PCDATA)>
<!ELEMENT Event (#PCDATA)>
<!ELEMENT Response (#PCDATA)>
<!ELEMENT Equipment (Name | Make | Model | SerialNumber |
DateTime |
    CurrentState | PreviousState | NetworkAddress |
    NetworkType | SoftwareRevision | ItemsInMachine
|
    ItemsInProcess | ItemsProcessed | CurrentRecipe |
    PreviousRecipe | AllRecipes | Alarm | ItemEnter |
    ItemExit | ItemProcessStart | ItemProcessEnd |
    RecipeChange | StateChange)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Make (#PCDATA)>

```

```

<!ELEMENT Model (#PCDATA)>
<!ELEMENT SerialNumber (#PCDATA)>
<!ELEMENT CurrentState (#PCDATA)>
<!ELEMENT PreviousState (#PCDATA)>
<!ELEMENT NetworkAddress (#PCDATA)>
<!ELEMENT NetworkType (#PCDATA)>
<!ELEMENT SoftwareRevision (#PCDATA)>
<!ELEMENT ItemsInMachine (#PCDATA)>
<!ELEMENT ItemsInProgress (#PCDATA)>
<!ELEMENT ItemsProcessed (#PCDATA)>
<!ELEMENT CurrentRecipe (#PCDATA)>
<!ELEMENT PreviousRecipe (#PCDATA)>
<!ELEMENT AllRecipes (#PCDATA)>
<!ELEMENT Alarm (#PCDATA)>
<!ELEMENT ItemEnter (#PCDATA)>
<!ELEMENT ItemExit (#PCDATA)>
<!ELEMENT ItemProcessStart (#PCDATA)>
<!ELEMENT ItemProcessEnd (#PCDATA)>
<!ELEMENT RecipeChange (#PCDATA)>
<!ELEMENT StateChange (#PCDATA)>
<!ELEMENT Charting (#PCDATA)>
<!ELEMENT Line (#PCDATA)>
<!ELEMENT Materials (#PCDATA)>
<!ELEMENT Operation (#PCDATA)>
<!ELEMENT Order (#PCDATA)>
<!ELEMENT ProcessPlan (#PCDATA)>
<!ELEMENT Product (#PCDATA)>
<!ELEMENT Report (#PCDATA)>
<!ELEMENT Resource (#PCDATA)>
<!ELEMENT Task (#PCDATA)>
<!ELEMENT WorkInstructions (#PCDATA)>

```

The NEMI Testbed-DTD file contains Element type declarations, which identify the names of elements and the nature of their content. A typical element type declaration in this DTD looks as follows:

```

<ELEMENT Command (Get | Set | Event | Response)>
    or
<ELEMENT Event (#PCDATA)>

```

The first declaration identifies the element named Command. Its content model defines what an element may contain. In this case, a Command can contain Get, Set, Event or Response. The vertical bar indicates an 'or' relationship. Declarations for Get, Set, Event, Response and all other elements used in any content model must also be present for an XML processor to check the validity of a document.

The second declaration `<!ELEMENT Event (#PCDATA)>` reflects this request, where, in addition to element name Event, the special symbol #PCDATA is reserved to indicate character data. The moniker PCDATA stands for parsed character data. With these encryption elements, the XML parser can parse and validate the following related NEMI XML message:

```

<?XML VERSION="1.0"?>

```

```

<!DOCTYPE NemiMessage SYSTEM "nemi v1b.dtd">
<NemiMessage>
  <Control>
    <Sender>MPM Equipment</Sender>
    <Receiver>Database</Receiver>
    <Receiver>MPM REC</Receiver>
    <Command>
      <Event></Event>
    </Command>
    <Type>Parametric Data</Type>
    <Status>Good</Status>
    <DateTime>19980805100431.20+0400</DateTime>
  </Control>
  <Data>
    <Equipment>
      <NetworkAddress></NetworkAddress>
    </Equipment>
  </Data>
</NemiMessage>

```

The NEMI XML message structure consists of two main parts. The first part is the 'control' or header section containing information about sender, receiver, command, type, status, and datetime stamp. The second part is the 'data' section containing the actual data to be sent as part of the message.

The XML message shown above follows this structure. The control section of this message indicates that a message is being sent from MPM Equipment to two destinations: Database and REC (Remote Equipment Console). The second part of this message contains the actual data. This data indicates that there is an Equipment Item Exit message for Board 10213 that is exiting Machine 1 (the piece of MPM equipment) with a Processing Complete event on August 5th, 1998 at 10:04 AM.

Another simple DTD (EquipMessage.dtd) which describes a Model of Equipment Processing States is shown below.

```

<!-- DTD version for Equipment processing state -->
<!ELEMENT EquipMessage (Machine, ProcessState)>
<!ELEMENT Machine (#PCDATA)>
<!ATTLIST Machine MachineID CDATA #REQUIRED>
<!ELEMENT ProcessState (Init|Idle|Production|Diagnostic|SubSetup)>
<!ELEMENT Init (#PCDATA) >
<!ELEMENT Idle (#PCDATA) >
<!ELEMENT Production (#PCDATA)>
<!ELEMENT Diagnostic (#PCDATA) >
<!ELEMENT SubSetup (#PCDATA) >

```

This DTD shows that there is an EquipMessage that is sent to a machine when it is in a certain processing state. The valid processing states for a piece of equipment are Init, Idle, Production, Diagnostic, or SubSetup.

```
<?XML VERSION="1.0"?>
<!DOCTYPE EquipMessage SYSTEM "MyDTD.dtd">
<EquipMessage>
  <Machine MACHINEID="GSM-1">
    </Machine>
    <ProcessState>
      <Production>In Production</Production>
    </ProcessState>
  </EquipMessage>
```

This message refers to the EquipMessage.dtd document type definition file shown above. It indicates that the process state of the equipment represented by the object GSM1 is In-Production.

5.4 Use of the HyperText Transfer Protocol (HTTP)

HTTP is the acronym for HyperText Transfer Protocol, the protocol for moving hypertext files across the Internet. It requires an HTTP client program on one end (a web browser), and an HTTP server program (a web server) on the other end. HTTP is the most important protocol used on the World Wide Web (WWW).

The common thread among all of these services is that they require a sequence of interactions between the user and a server (or servers). In most cases, all interactions are typical HTTP commands such as Get, Post, and Connect.

More sophisticated service delivery applications involve highly interactive client side processing as well as a means for the client to communicate with the server over the Internet. Distributed objects provide a particularly useful mechanism for this client-server communication.

The equipment interface in the NEMI plug and play framework adheres to HTTP and XML for exchanging messages between clients and servers. Adopting HTTP and XML is also a good proactive step to accommodate the evolution of Internet technologies. All plug and play devices and applications must have the ability to be self-describing and allow automatic configuration. When plugged into the framework, these business objects must announce their presence using a simple discovery protocol based on the HTTP protocol and immediately be ready to share their services with other business objects that request them.

5.5 Building a plug and play framework

The NEMI plug and play framework provides a common set of interfaces for accessing devices and application services, enabling the operational unification of diverse business objects.

The NEMI plug and play framework is a three-tiered architecture, as shown in figure 6. The goal is to build a message-based remote service / method invocation mechanism based on HTTP-XML. Browser-based clients using Dynamic HTML (DHTML) or XML communicate with the HTTP web server via XML. The web server in turn sends XML messages to the XML enabled middleware. The XML enabled middleware takes care of routing the messages to other business objects written in C++, Visual Basic, or Java. It can also send the XML message to

either an object-oriented or relational database. We chose the message-based framework because this is the best way to become language-independent and totally detached from the decision about which component object model to use: the Object Management Group's Common Object Request Broker Architecture (CORBA) or Microsoft's Component Object Model/Distributed Component Object Model (COM/DCOM). If the framework mechanism is pure XML message-based then it can accommodate both CORBA and COM/DCOM implementations. CORBA objects can run on Unix servers and COM objects under Microsoft Transaction Server (MTS). There is no need to be bound to a single component object model.

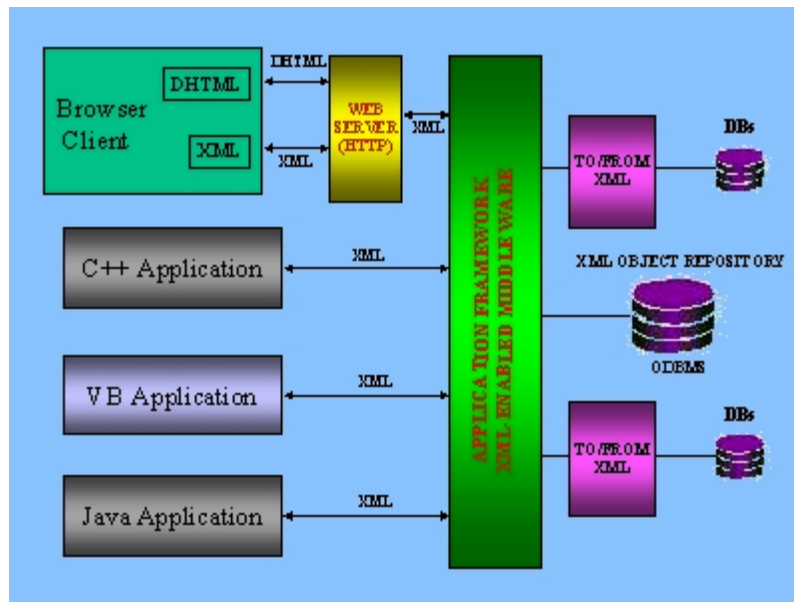


Figure 6. Message-based framework architecture.

However, this heterogeneous message-based environment requires that transactions exist across multiple components. The object models need to be very well coordinated. Currently there are no adequate tools to overcome these limitations in the way transactions are handled in this kind of structure.

Another possibility is to look at directory services such as Novell NDS, Microsoft Windows 2000 (NT5) Active Directory and Oracle Internet File System. They might provide another viable remote service and method invocation mechanism without messaging. A directory service, at least in part, is an object-oriented database representing network users, applications and resources. Within each object is stored specific information about the individual user or resource found on the network. Objects are structured hierarchically in a directory tree providing a framework that reflects a plant-floor operational organization. The NEMI plug and play project chose not to use directory services due to the volatility of this technology.

In addition, RMI and Jini in the Java world address the same problem where Jini, built on top of RMI, provides a Discovery and Lookup mechanism for the registration, discovery and interaction of network services. This technology is currently being investigated as part of this project

The proof-of-concept for the proposed message-based architecture has been implemented and demonstrated several times, using the NEMI plug and play framework testbed implementations located at the Georgia Institute of Technology. The NEMI plug and the

evolution of the architecture and applications developed as part of this project are described in the next section of this paper.

6 *NEMI plug and play factory implementation*

6.1 *NEMI plug and play testbed description*

Since the purpose of the NEMI plug and play project is to develop a framework that can be used across a geographically dispersed enterprise, the testbed made use of equipment and developers located throughout the United States. The various components that made up the testbed were connected via the Internet and information flowed between the components to simulate an organization that was manufacturing products at various locations. Portions of the testbed were located at AMP Incorporated (Harrisburg, PA), Electronic Data Systems (Kokomo, IN), Georgia Institute of Technology (Atlanta, GA) and Universal Instruments Corporation (Binghamton, NY). During the various demonstrations of the testbed activities, information flowed among the components of this virtual enterprise testing the feasibility of transferring live manufacturing data over the Internet. Testing this information flow assisted in flushing out challenges associated with a framework of this type. A depiction of the information flow during the demonstration conducted in Portland, Oregon can be seen in figure 7.

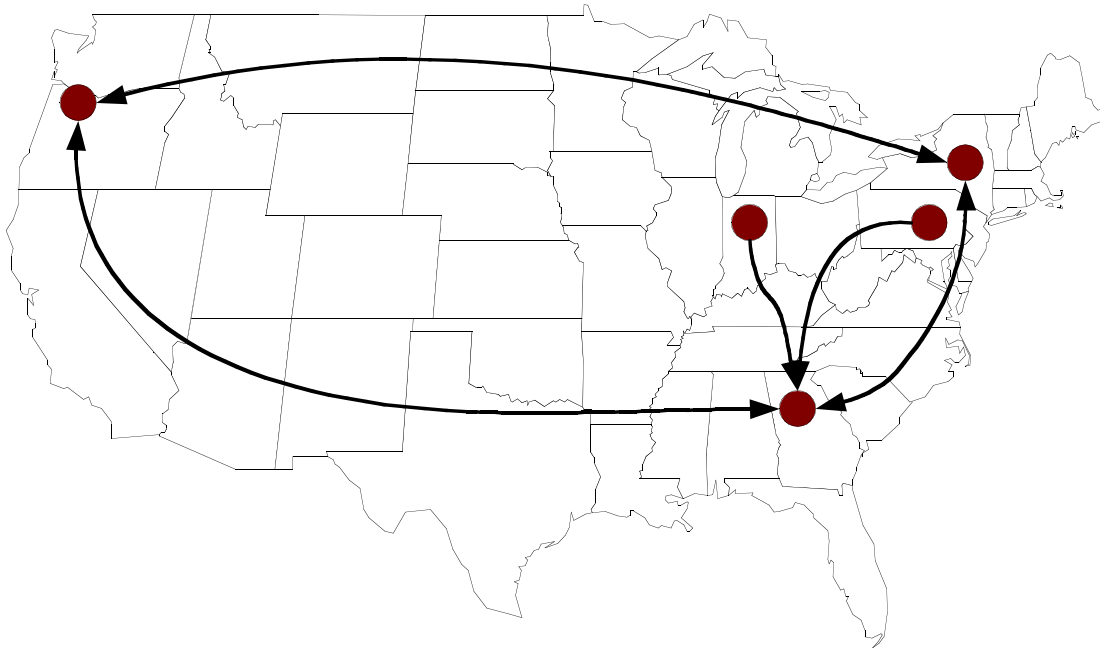


Figure 7. Typical communication paths during demonstrations.

Although the testbed made use of resources located at various locations, the heart of the testbed was located in the Center for Board Assembly Research (CBAR) electronics assembly lab at the Georgia Institute of Technology in Atlanta, GA. The CBAR lab contains a state-of-the-art high-volume electronics manufacturing line contained in a class 100,000 clean space. All the equipment is conveyORIZED and is GEM compatible. The equipment located on the line consists

of an MPM Ultraprint 3000 Screen Printer, a Cyber Optics Sentry 2000 inspection machine, a Camalot System 3800 adhesive dispenser, Siemens Siplace 80 S20 and Siplace 80 F5 placement machines, an MV Technologies GS-1 optical inspection machine, an Electrovert Omniflow 5 reflow oven, and a GenRad 2281 in-circuit tester. figure 8 shows a schematic representation of the line. A photograph of the actual line is shown in figure 9.

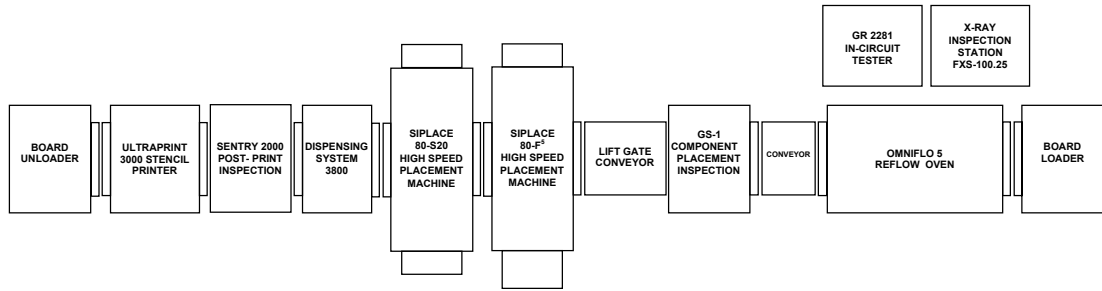


Figure 8. Schematic representation of the NEMI plug and play testbed line.



Figure 9 . Photo of the NEMI plug and play testbed line.

6.2 *NEMI plug and play testbed development*

From January 1998 until March 1999 the testbed investigated four different framework configurations. At the conclusion of each development cycle, the lessons learned were transferred to the framework definition task group, and a demonstration was conducted to showcase the software applications developed for use on the testbed. Table 2 lists the four framework technologies that were investigated through each iteration of the investigative cycle. The fifth version is the current version of the framework that was under investigation as of this writing.

6.2.1. *Testbed version one.* The first framework configuration that was installed on the testbed was the Product Realization Environment (PRE) which was developed by Sandia National Laboratories. PRE is a lightweight, horizontal, CORBA (Common Object Request Broker Architecture) based framework that was developed at Sandia National Laboratories and is used by the United States Department of Energy. PRE allows the exchange of data among objects and remote applications through its environment. It is hardware, platform, and language independent and contains Application Programming Interfaces for interacting with data and application objects through seven interfaces and 35 methods. PRE also provides framework services such as security, data persistence, object communication, and trading services. The goal of PRE is to provide a powerful, yet easily learned framework based on an industry standard (CORBA).

The manifestation of PRE on the testbed is shown in figure 10. PRE was used as a middleware to exchange information on a factory floor.

Although the PRE framework provided the technology needed to support the plug and play project, it was apparent that there would never be wide spread adoption of PRE. In this respect, PRE suffered from the same limitations as GEM.

Table 2. NEMI plug and play testbed tasks.

<i>Version</i>	<i>Technologies</i>	<i>Development Period</i>	<i>Demonstration</i>
1	Product realization environment	January 1998 to March 1998	March , 1998 NEPCON - West Anaheim, CA, USA
2	TCP Sockets, JDBC	April 1998 to May 1998	May , 1998 NEMI Technical Committee Meeting Portland, OR, USA
3	TCP Sockets, XML, JDBC	May 1998 to October 1998	October , 1999 IPC/SMTA Expo Providence, RI, USA
4	HTTP, TCP Sockets, XML, JDBC	October 1998 to March 1999	February 24, 1998 NEPCON - West Anaheim, CA, USA
5	JINI, HTTP, TCP Sockets, XML, JDBC	March 1999	TBD

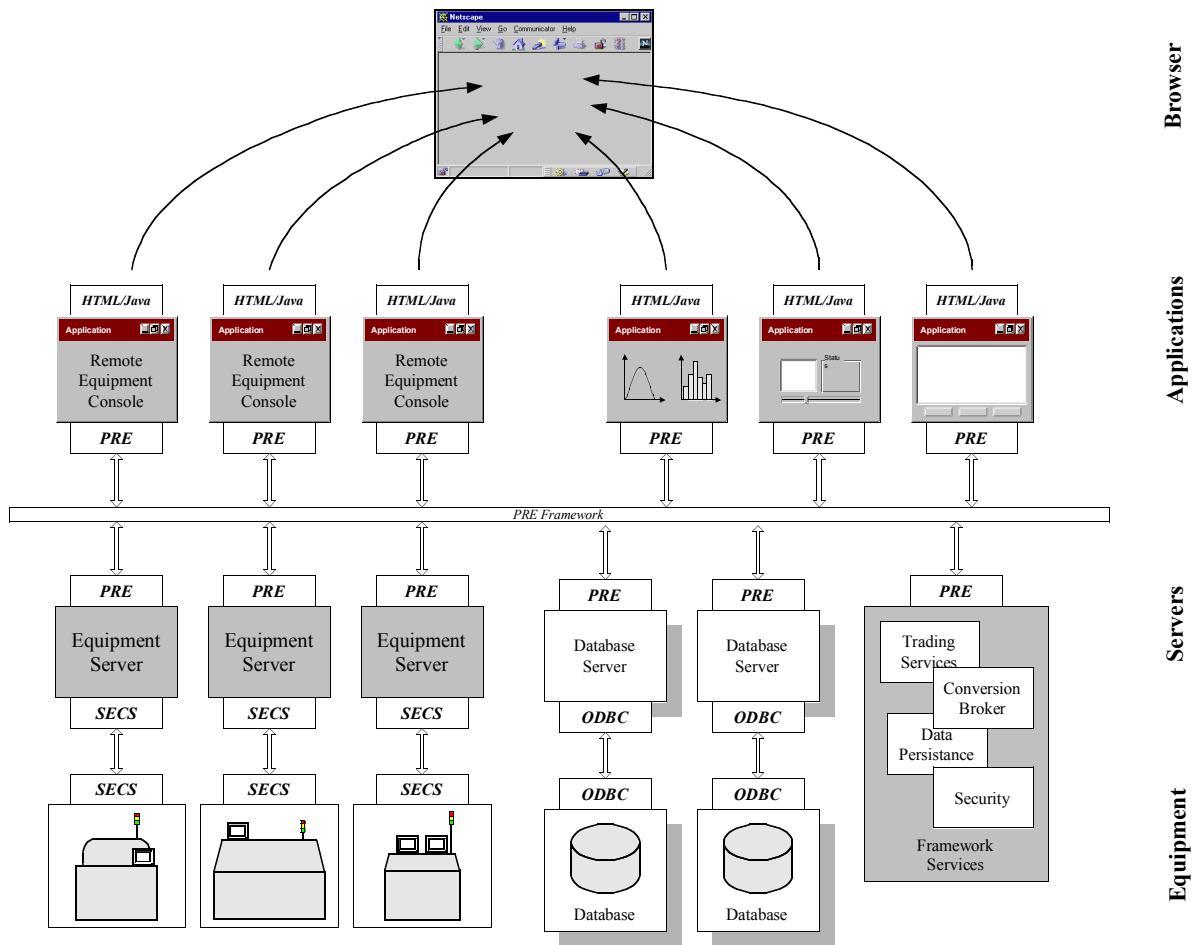


Figure 10. Testbed version one, making use of PRE.

6.2.2. *Testbed version two.* The second version of the framework made use of TCP (Transmission Control Protocol) sockets and JDBC (Java Database Connectivity). TCP sockets are the fundamental communication service used by a variety of programs such as: file transfer protocol (ftp), electronics mail, and web servers. Socket communication can be implemented through almost any language (e.g. C++, Java, or Visual Basic), and is hardware and operating system independent. JDBC provides an object-oriented, consistent access to databases through the Java programming language.

In this version of the framework, applications were configured as both socket servers and clients. The client portion of an application connected to the server side of another application and exchanged strings through the resulting socket connection. Data was placed in and retrieved from databases through a JDBC interface. Middleware services were not considered in this version.

The lack of a middleware component greatly restricted the plug and play nature of this version. Without middleware, message structures and contents and equipment-application connections would be very inflexible.

6.2.3. *Testbed version three.* The third framework that was investigated on the testbed made use of TCP Sockets, XML and JDBC. Figure 11 provides an overview of this version. The main feature of this development effort was the middleware depicted in the center of figure 11. Applications connected to the middleware through a predetermined TCP socket port, and passed XML messages to the middleware. The middleware parsed the XML message and directed it to the appropriate application based upon information contained in the message. JDBC connectivity was used independent of the middleware to exchange information with the framework databases.

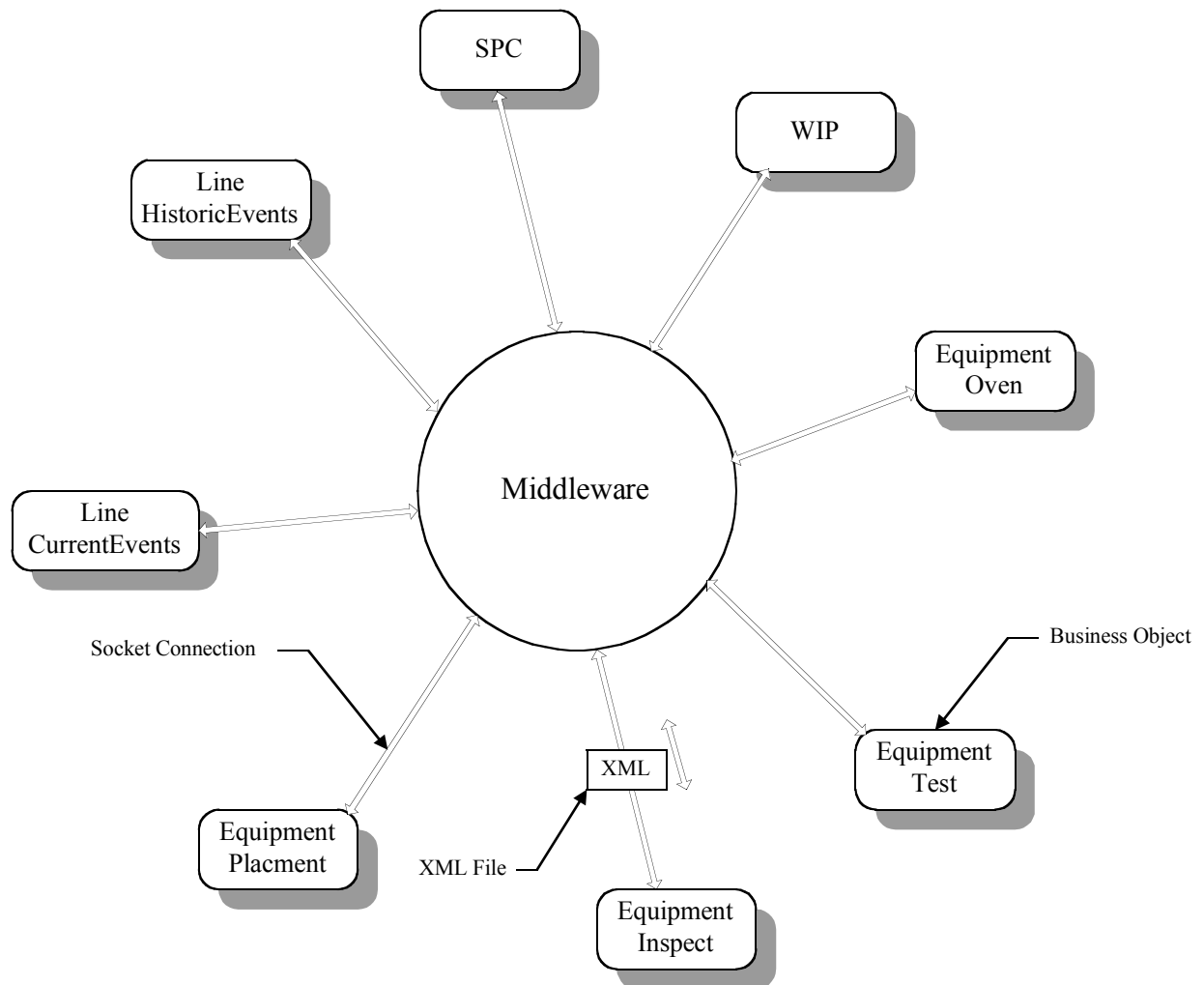


Figure 11. Testbed version three, making use of sockets, XML and JDBC.

6.2.4. *Testbed version four.* The fourth version of the framework made use of HTTP, XML messages, TCP sockets and JDBC. Figure 12 shows a generic version of this framework implementation. The key feature of this implementation is that HTTP is used as the middleware. HTTP is the communication protocol used by web browsers and web servers. Each object within the framework makes use of a web server as an interface to the other objects contained in the framework. If information is to be exchanged between objects, a connection is made via sockets

in the same way that web browsers connect to web servers. Once the connection has been made, HTTP commands with embedded XML messages are exchanged among the objects. This version of the testbed makes use of a distributed technique to exchange information in contrast to the centralized configuration investigated in the previous version. In order to provide a more detailed explanation of this configuration, the various components that were developed and demonstrated will be reviewed.

Version four of the framework addressed most of the requirements specified by the framework definition team; however, it does not provide a mechanism to support automatic detection of newly added equipment and applications.

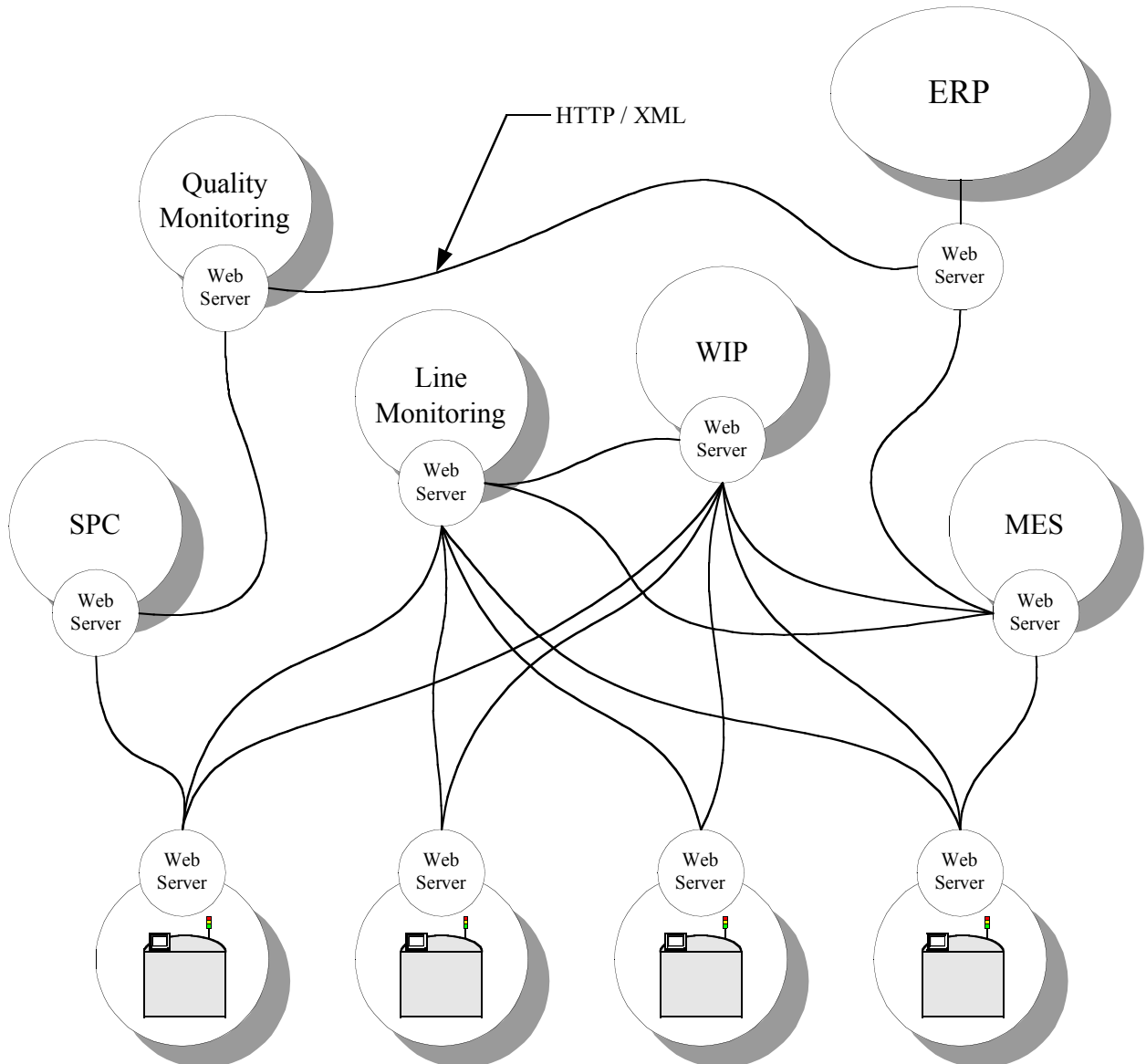


Figure 12. Testbed version four, making use of HTTP, XML, sockets, and JDBC.

6.3 NEMI plug and play testbed applications

6.3.1. *Factory console.* Since one of the basic premises of the framework is to be "web enabled," the testbed demonstrations were conducted over the Internet. The portal into the framework was through a web page, which is called the factory console. The factory console contains an image map that is shown in figure 13. The web page also contains Javascript code, which provides functionality to the image map. When one of the buttons is clicked on the image map, a web page is launched that contains an applet. The applet provides a graphical user interface to the framework application. The lines on the console connecting the various icons on the image map indicate communication paths between the objects. These are the same communication paths that are shown in figure 12, where XML messages are exchanged using HTTP.

The images of the equipment shown on the console represent the equipment used on the testbed. Although the equipment is located at various geographical locations, the information associated with the equipment can be accessed from this single web page. The buttons represent framework applications, and the camera icon provides a video feed from the testbed.

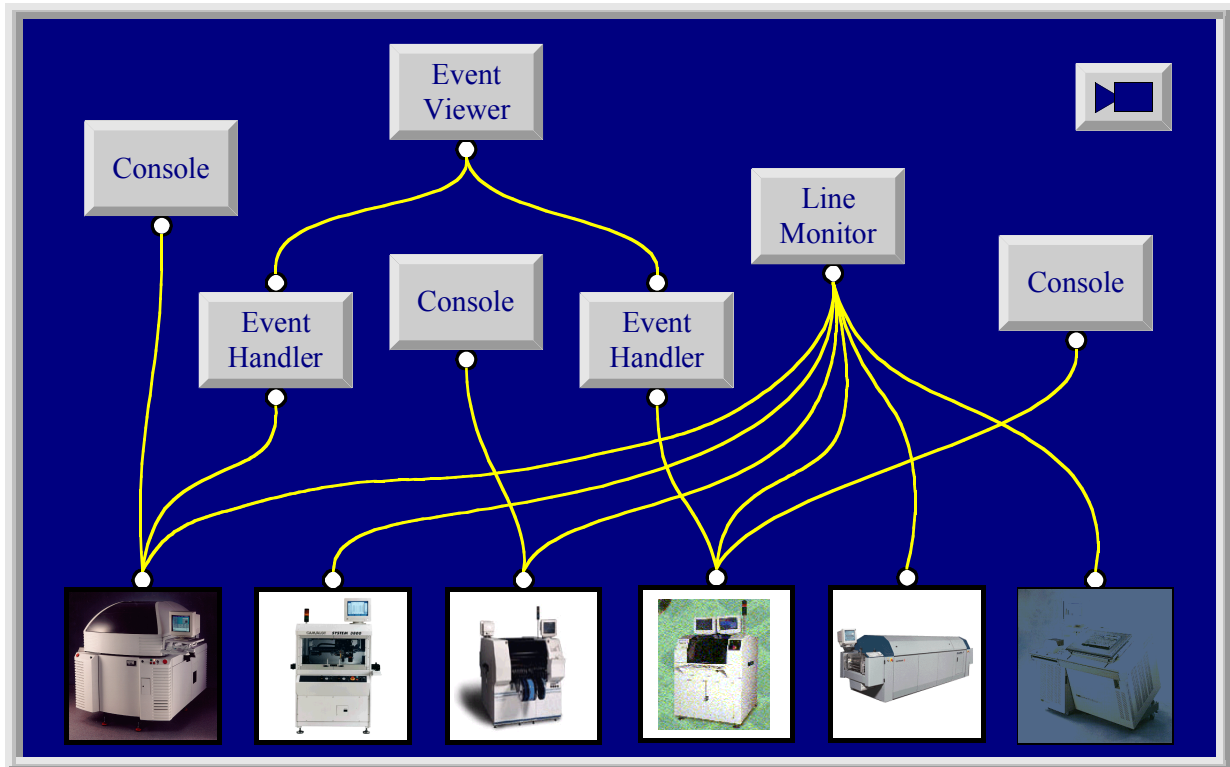


Figure 13. Factory console used on testbed framework version four.

6.3.2. *Remote equipment console.* The buttons labeled “console” launch a remote equipment console (REC). A typical REC is shown in figure 14. An REC connects to an equipment object and allows a user to interact with the equipment. When an REC is launched, an REC applet graphical user interface is presented. A user establishes an HTTP connection to the equipment web interface by clicking the “connect” button. The appropriate data type of interest is selected from the drop-down menu and then the “send” button is clicked. An XML message is developed and sent to the equipment web server. The equipment receives the message and sends a reply. The XML message received back from the equipment object is parsed, and the appropriate data is displayed. Both the outgoing and incoming XML messages are also displayed. Information such as the machine’s current state, time and configuration data can be determined through this manner.

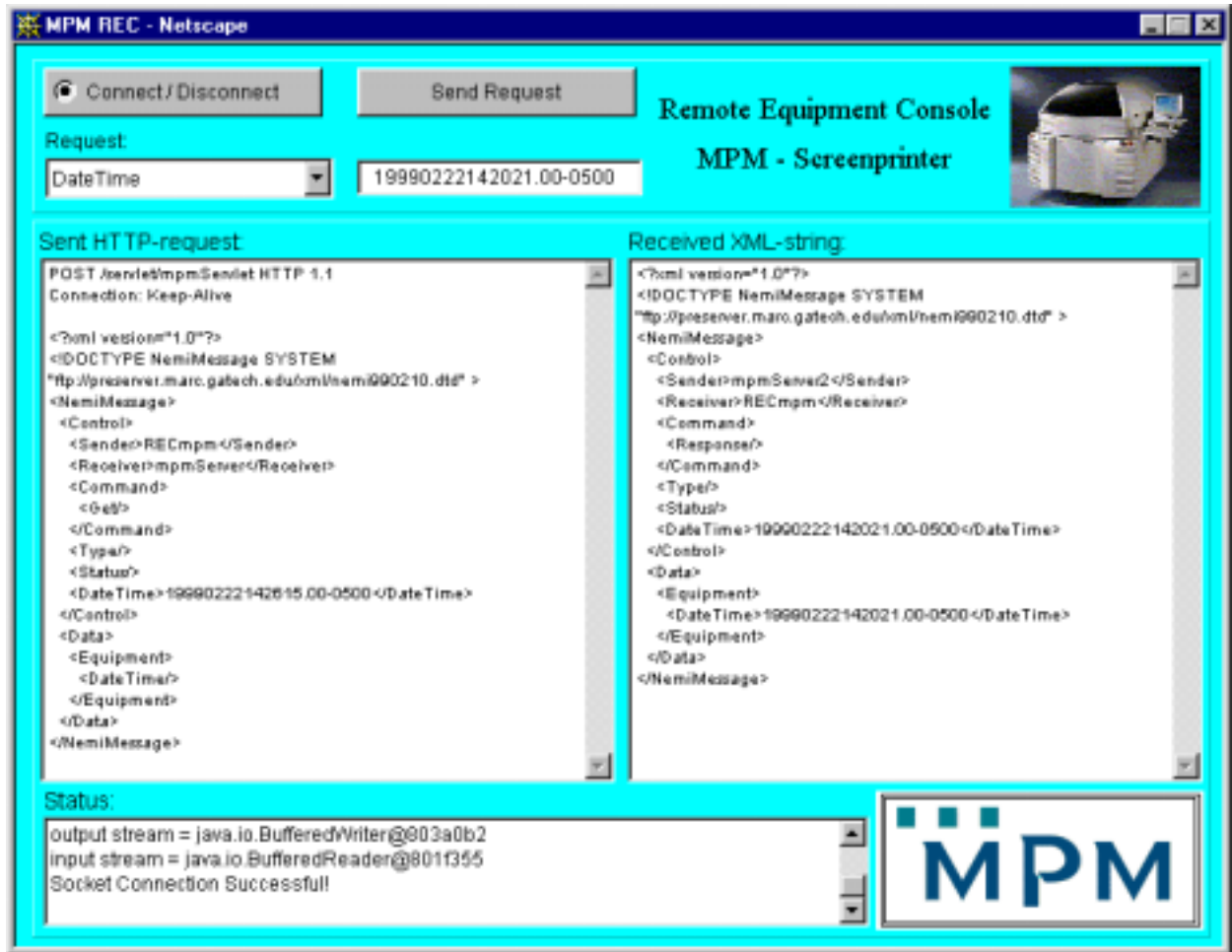


Figure 14. Remote equipment console graphical user interface.

6.3.3. *Event handler.* The buttons labeled “event handler” launch a web page that contains an event handler applet associated with a particular piece of equipment. The event handler (figure 15) is a framework application that receives asynchronous messages from the equipment object, and passes them to a database. When the event handler is launched, the user connects to the equipment web interface and the database by clicking on the appropriate buttons. An HTTP connection is made to the equipment object, and a JDBC connection is made to the database. XML messages are received via the HTTP connection, the event handler parses the XML message and sends the information to the database via structured query language (SQL) over JDBC. The event handler displays the XML message received from the equipment

object, and updates a table with parsed data. Examples of asynchronous events that are received from the equipment object include board enter, board exit, processing start and processing end.

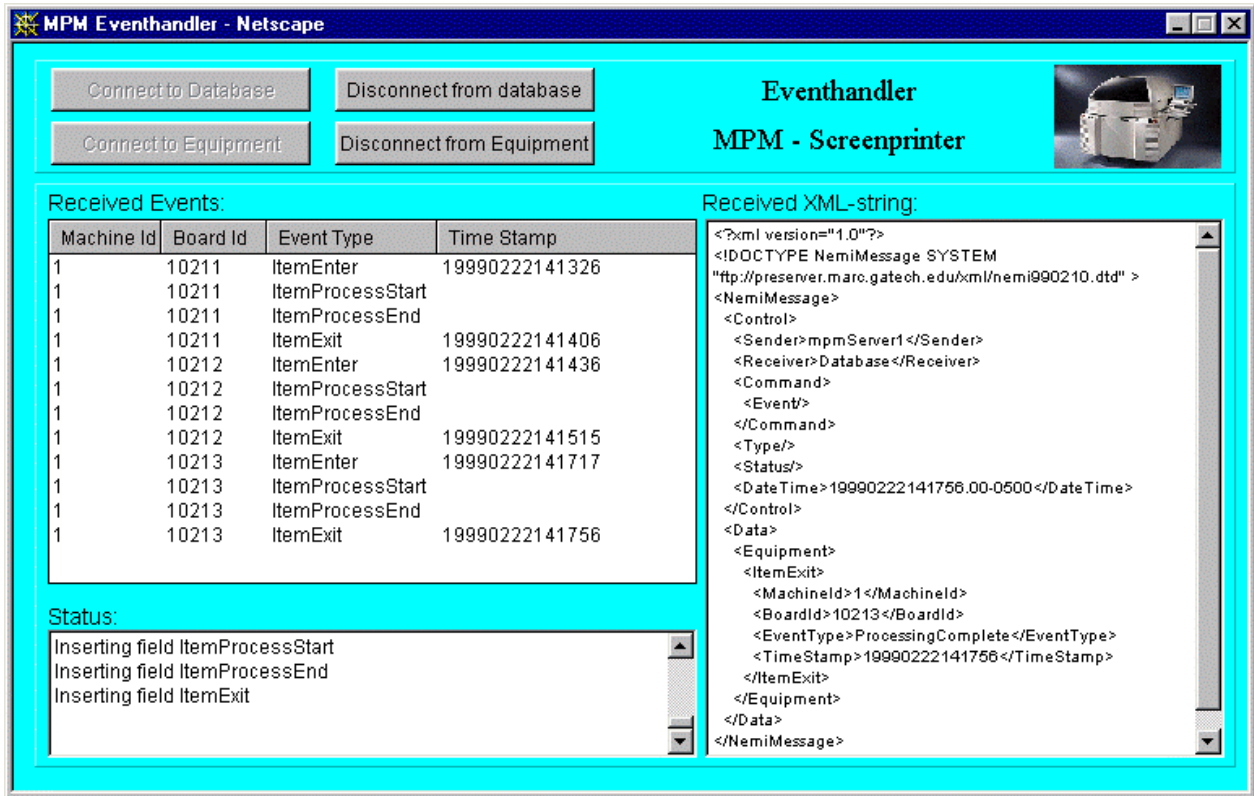


Figure 15 . Event handler graphical user interface.

6.3.4. *Event viewer.* Another framework application that can be launched from the factory console is the “event viewer” (figure 16). The event viewer allows a user to view machine event data that has been stored in the database by the event handler applications. Machine events received from any of the equipment on the testbed can be displayed using the event viewer for any time period of interest. The event viewer connects to the framework database via JDBC and queries the data structure through SQL. The unique aspect of this application is its ability to provide data about any piece of equipment, regardless of geographic location or vendor. This application demonstrates the framework interoperability.

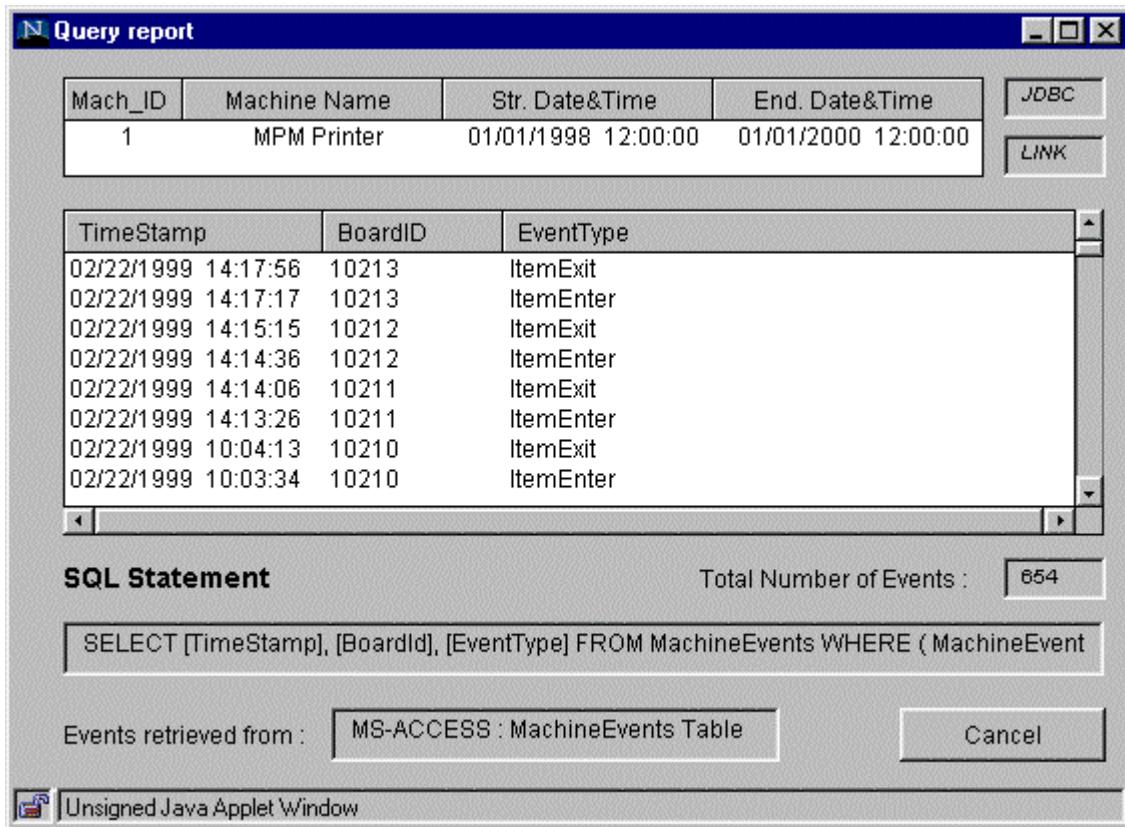


Figure 16. Event viewer graphical user interface.

6.3.5. *Line monitor.* The line monitoring application is another example of an interoperable application (figure 17). It connects to multiple machine equipment objects and receives asynchronous events from the machines. The application is a graphical representation of the current states of the machines connected to the framework. As printed circuit boards move through the line, the line monitoring images are updated to reflect the current state of each machine. The application contains multiple applets on a single web page. Each applet connects to the appropriate equipment object through an HTTP connect. XML messages are received from the equipment objects and parsed by the applets. The line monitoring application can send and receive both synchronous and asynchronous data from equipment produced by different vendors.

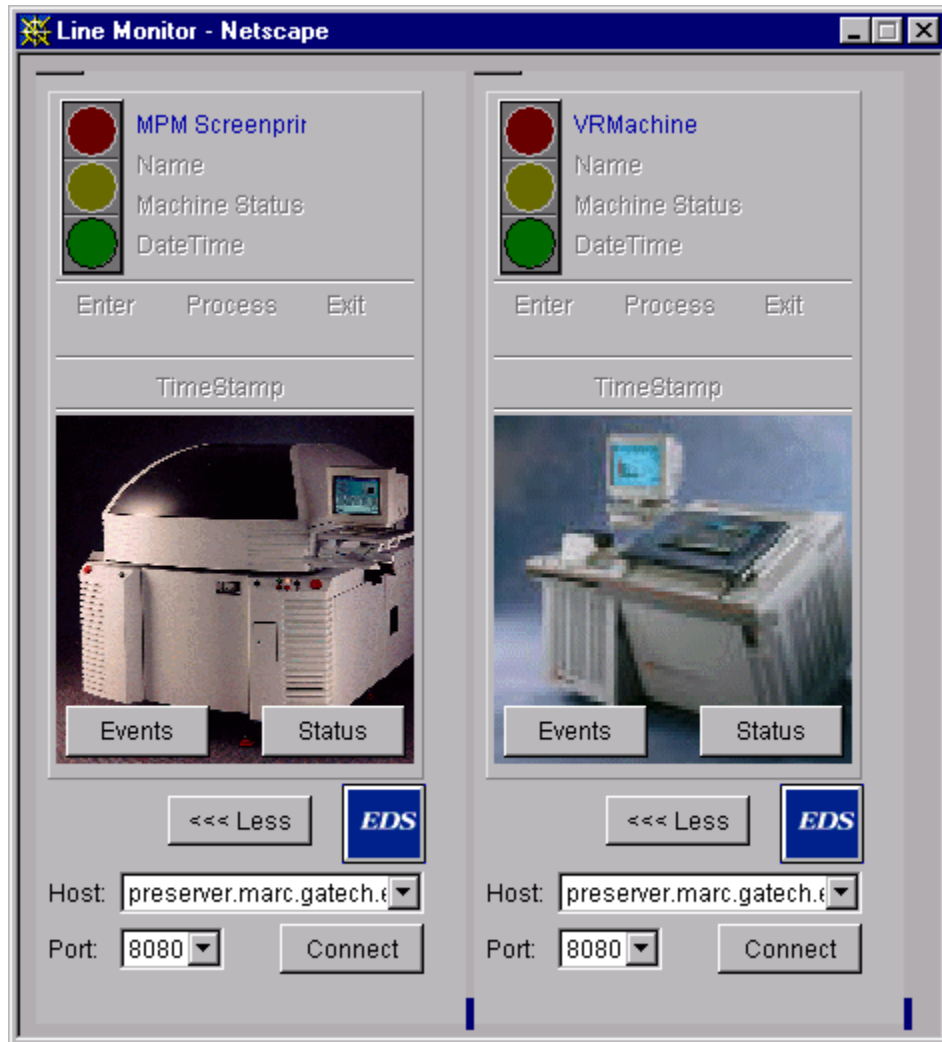


Figure 17 Line monitoring graphical user interface.

7. Conclusions and Future Work

The National Electronics Manufacturing Initiative's (NEMI) plug and play factory project addressed several issues relating to how to quickly integrate new pieces of electronics assembly equipment and software into a shop floor line management system. The necessary technical infrastructure was designed, developed and demonstrated over a two-year period.

The NEMI plug and play framework implementation involved several elements, including data represented via business objects, a set of defined application services, a message-based middleware infrastructure using HTTP and XML, and a set of equipment web servers.

The project implementation results demonstrated that the project goals of reduced equipment and software integration time and cost could be achieved with currently available technologies.

The current focus for the project is to get these new standards adopted by industry. In order to reach that goal NEMI and the project team have begun working with the IPC, an ANSI

accredited standards organization, to promote wider industry participation and to begin the standardization process for the NEMI plug and play factory standards.

The pace of technological change continues to accelerate. Two new technologies that did not even exist when this project began two years ago now seem like they may be of great utility in the future. Jini handles the issues of publish and subscribe for devices and applications in a Java environment. The applicability of this technology to the printed circuit board domain is currently being investigated. The current XML-momentum in the industry will open new ways for further improvements in form of new standards. The XML metadata interchange format (XMI) specifies an open information interchange model that is intended to give developers working with object technology the ability to exchange programming data over the Internet in a standardized way, thus bringing consistency and compatibility to applications created in collaborative environments. The proposed standard will allow developers to leverage the web to exchange data between tools, applications, and repositories to create secure, distributed applications built in a team development environment. This type of capability should make it even easier in the future to design and implement solutions that exhibit plug and play characteristics.

Bibliography

Beveridge, J., and Wiener, R., 1997, *Multithreading Applications in Win32*, (Addison-Wesley).

Flanagan, D., 1997, *Java in a Nutshell*, (O'Reilly and Associates).

<http://daytona.ca.sandia.gov/pre/>

<http://isd.cme.nist.gov/info/omacapi>

<http://www.csee.umbc.edu/lait/projects/ciimplex/>

<http://www.niiip.org/index.html>

<http://www.openapplications.org/>

<http://www.w3.org/Protocols/>

<http://www.sematech.org/public/division/fi/cim/cimhome.htm>

<http://www.w3.org/XML>

<http://www.w3c.org/Protocols/>

Hunter, J., and Crawford, W., 1998, *Java Servlet Programming*, (O'Reilly and Associates).

MESA International, 1997, MES Functionality & MRP to MES Data Flow Possibilities, White Paper Number 2.

Object Management Group, 1998, CORBA-based Machine Control White Paper, OMG Document mfg/98-02-03.

Open Applications Group, 1998, Middleware API Specification, OAMAS -1.

Powell, T., 1998, *Web Site Engineering*, (Prentice Hall PTR).

Quinn, B., and Shute, D., 1995, *Windows Sockets Network Programming*, (Addison-Wesley).

SAP AG, 1997, Benefits of the Business Framework.

Semiconductor Equipment and Materials International, 1995, Equipment Automation/Software Volume 1.

Semiconductor Equipment and Materials International, 1995, Equipment Automation/Software Volume 2.

St. Laurent, S., 1998, *XML: A Primer*, (MIS: Press).

Stevens, R. W., 1998, *Unix Network Programming*, (Prentice Hall PTR).

Swanton, B., 1998, Are Electronics Plants Outgrowing Their Software?, AMR Research.

Wagner, R., 1997, *JavaScript Unleashed*, (Sams.net).