MITEM Host Adapter

Host Integration to
TIBCO Rendezvous and Active Enterprise

IBM Mainframe
AS/400
DEC VAX
Data General
Tandem
Unisys
Bull
Hewlett Packard
Unix
ICL

Published by:

MITEM Corporation

info@mitem.com

www.mitem.com

Tel: 800-82-MITEM

Last update: July 1, 2002

TABLE OF CONTENTS

SECTION ONE:

OVERVIEW

1. Introduction

Established in 1985, MITEM is a provider of legacy integration software and e-Business solutions for Global 2000 companies and government entities. MITEM defines "the last mile" as the chasm between host systems and modern EAI platforms (TIBCO). MITEM has focused on delivering solutions to bridge this last mile and is recognized as the leader in legacy integration software.

MITEM software has been deployed in diverse industries such as public utilities, financial services, manufacturing, health care, education and government. MITEM's global customer base includes ABN AMRO, GE/ERC, Harley-Davidson, Con Edison, London Electricity, American Electric Power, Bayer, Swiss Life, Lockheed Martin, CalPERS, US Postal Service and US Navy.

MITEM Corporation, in joint development with TIBCO Software, offers a Host Adapter for TIBCO Rendezvous and Active Enterprise. This Host Adapter provides screen-based integration to the more than 300,000 applications that currently reside on legacy mainframe and midrange systems.

The MITEM Host Adapter complements the existing adapters offered by TIBCO, as indicated in the table below. This document introduces the MITEM Host Adapter, which is a new type of adapter to address a common approach to integrating TIBCO products with mainframe and midrange systems.

TIBCO Adapters	**Offered by**	**Examples**
1. Packaged Applications	TIBCO	Siebel, PeopleSoft, Vantive, Clarify, SAP
2. Databases	TIBCO	Oracle, MS SQL, Sybase, ODBC
3. Network Technologies	TIBCO	COM, CORBA, MSMQ
4. Custom Developed	TIBCO	Via TIBCO's Software Development Kit
5. Host Applications	**MITEM**	**IBM Mainframe, AS/400, DEC VAX, Tandem, Data General, Unix, Unisys, Bull, ICL, Hewlett Packard, …**

TIBCO recognized the need for an efficient method to integrate Rendezvous and Active Enterprise with these legacy host systems, and researched the best approach to this challenge. It was not practical to build an adapter for thousands of custom host applications. Instead, TIBCO searched for a software tool that would provide a repeatable methodology of integration to screen-based host applications.

After an extensive evaluation TIBCO chose MITEM as their preferred technology partner for legacy host integration. The MITEM technology is robust, mature, and well proven to deliver a simple solution to the complex environments of mainframe and midrange systems. MITEM's signature product, Mitem*View*, is the core message processing system for the Host Adapter.

The Host Adapter interfaces with any IBM mainframe, AS/400, DEC VAX, Tandem, ICL, Data General, Unisys, Unix, or Hewlett Packard host computer. In the list below are just a few examples of system configurations where the MITEM Host Adapter would apply. Environments and applications supported include:

Adabas	Conversys	Metavante
ACP	CSG	Misys
ADP	Cullinet	Natural
ADSO	Datacom/DB	OS/400
Alltel	FiServ	PL/1
Amisys	Geac	RPG
ASI	Hogan	TPF
BPCS	Ideal	VSAM
CICS	IDMS	
Cincom	IMS	And thousands of
Cobol	IMS/DB	others…..
CoolGen	IMS/DC	
Life-70	Mantis	

Mitem*View* closely parallels TIBCO's own Rendezvous and Active Enterprise hallmark messaging characteristics. The Host Adapter's low cost of ownership also makes it a viable alternative to a custom adapter approach. The Host Adapter's characteristics include:

- High performance (the speed of the mainframe or midrange system's messages to/from Rendezvous or Active Enterprise)
- Asynchronous
- Event-driven
- Support of reliable and certified messaging
- Accelerated implementation

2. A World with 300,000+ Applications

We have already established that there are more than 300,000 different host applications still running on mainframe and midrange systems. These include financial applications, ERP systems, insurance processing applications, banking systems, government-specific applications, and a realm of every possible application imaginable. These host applications are classically accessed via screen-based terminal emulation software. For example, in the IBM mainframe world, terminal emulation is via 3270; for Unisys, it could be T27 or UTS-60, and so on.

Many of these 300,000 applications have been customized or home-grown to suit the specific organization's needs. Unlike the modern client/server applications like SAP, Siebel or PeopleSoft, the mainframe or midrange systems typically do not have API's or open architectures, making integration with products like TIBCO challenging at best.

These two mainframe screens are from an IBM mainframe Purchase Order application. To integrate to the business logic of these types of host systems requires an integration environment that allows you to manage each and every 'host state', associated with the application. There may be 100's or 1,000's of host states for every application.

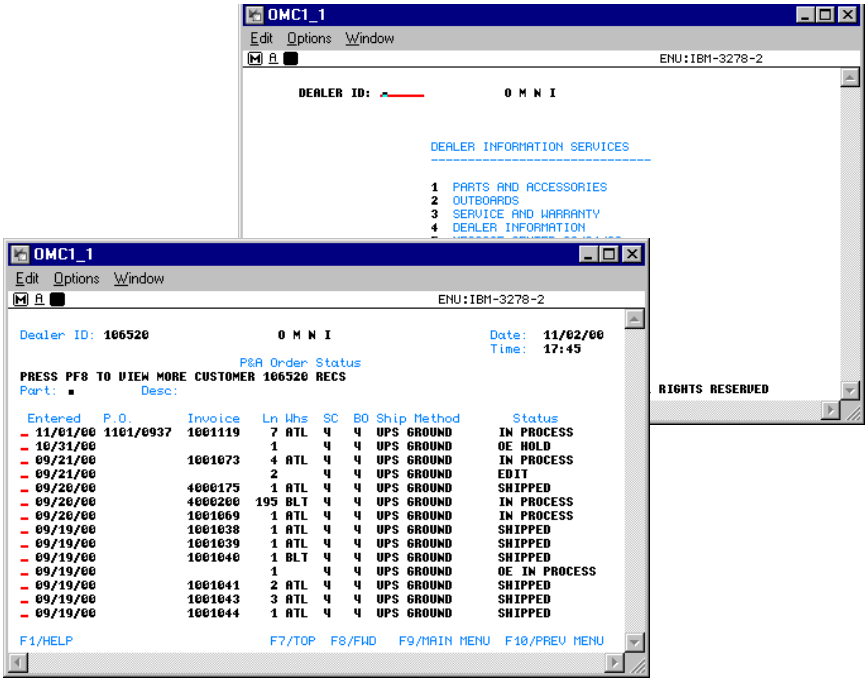


Figure 1: Sample mainframe screens from a 3270 terminal emulation window.

As an example, take a TIBCO supply chain event that requires an update to an inventory transaction on a host application, where the business logic determines which components are associated with a specified assembly. First, the host application must be checked to query the assembly component listings, then traverse through multiple screens to view the specific part that requires updating. The update can occur, and the application will advise the user if the transaction was completed or not. Error handling will occur along the way, in accordance with the edits set forth in the business logic of the host application. This example could require access to 10 – 20 screens.

Taking this Purchase Order application as an example of those thousands of host applications, how does one address this integration requirement to the business logic of the application? Because applications like this typically have been heavily modified, there is no generic adapter that is feasible. It is impossible to deliver a “no-coding” approach to integration to these mainframe or midrange systems because of the complexity of these environments.

The optimal solution provides:

- An event-driven (non-procedural) approach to processing the “screen” messages, greatly minimizing the coding effort
- A non-invasive integration to the host system (the host application is untouched and no additional software on the host is required)
- A tool kit that is easy to use, yet delivers functionality to handle the most complex integration needs
- A seamless integration of the mainframe or midrange system’s application to Rendezvous or Active Enterprise.

The Host Adapter from MITEM is the solution.

3. The Host Adapter from MITEM

TIBCO Adapters enable packaged applications, databases and other technologies to be active participants in the enterprise information flow. Adapters allow new components to be added to the system quickly and easily, without disturbing the existing infrastructure. Additional message transformation and business process automation can be handled once the data is published to the network, but adapters isolate the application from those more complex actions.

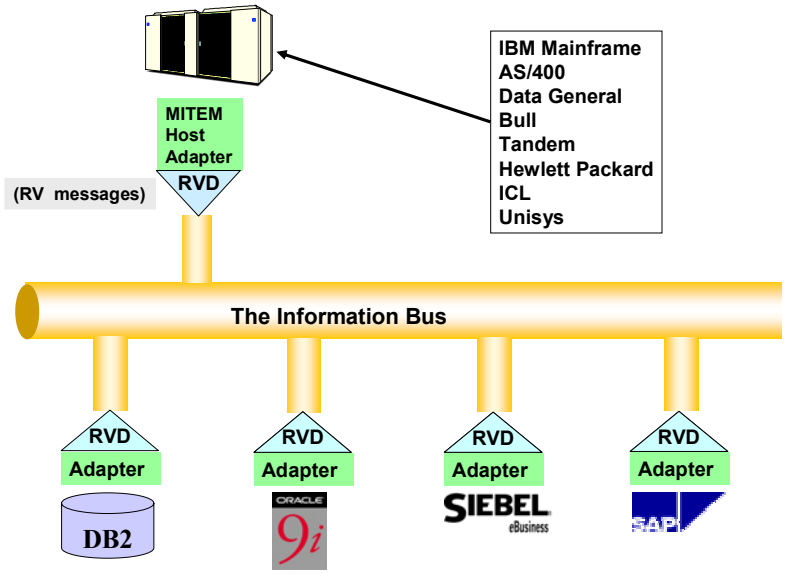


Figure 2: The MITEM Host Adapter is used to provide the last mile connection between The Information Bus and myriad host systems.

We established in the previous section that it would be impossible to create thousands of adapters that would provide out-of-the-box integration to all of the applications running on the IBM mainframe, AS/400, Data General, Bull, Tandem, Hewlett Packard, ICL and Unisys systems. As a practical alternative, the Host Adapter provides message brokering to and from these host systems. The Host Adapter, utilizing MITEM's signature integration software, MitemView, provides integration without modifying the legacy applications to message-enable them. This simplifies the integration control logic and maintains a flexible, loosely coupled architecture that enhances adaptability.

The MITEM Host Adapter:

- Generates the foundation framework for integrating the TIB with the host system(s) via a Wizard-driven configuration tool called ProStart
- Provides a streamlined way to identify, capture and manage the data stream (messages) from the screens of the host system
- Processes the request or reply from the Rendezvous or ActiveEnterprise message, and triggers and performs the necessary integration to the mainframe or midrange application(s).

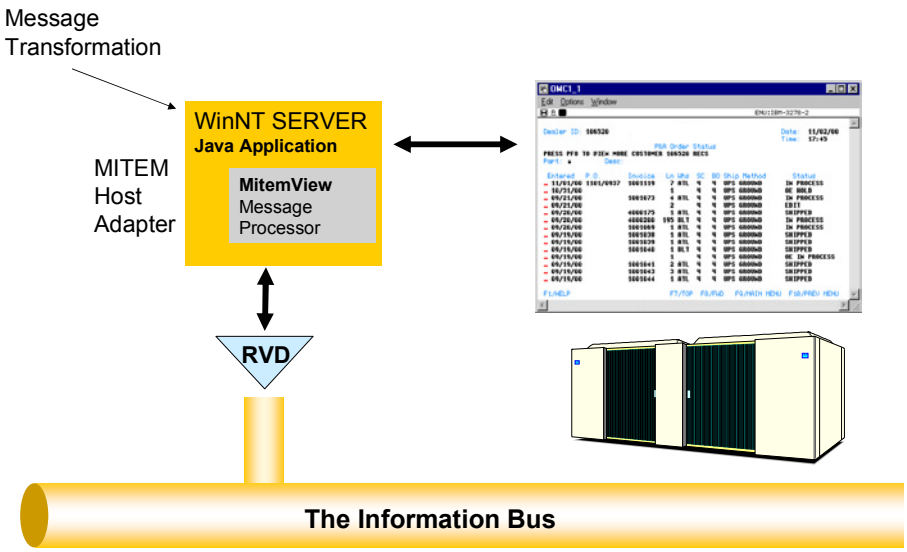


Figure 3: The Host Adapter listens for Rendezvous or Active Enterprise messages, and provides transformation of those messages to and from the host, returning the applicable message back to the TIB

In a typical scenario, the Host Adapter:

- Listens for a Rendezvous or Active Enterprise message
- Transforms it into structured data for use by the Java application
- The logic within the Java code, based on the RV message subject name, invokes the appropriate event to trigger integration to the host
- Based on pre-developed mappings to the specific host states or screens for that event, the Mitem*View* message processor handles the appropriate messages to or from the host
- If data is to be returned to the TIB, the Host Adapter transforms the host messages to a Rendezvous or Active Enterprise format and posts it to the TIB.

SECTION TWO:

TECHNICAL SUMMARY

4. Mitem*View* – The Integration Server

4.1 Introduction

At the heart of the Host Adapter for TIBCO is the Mitem*View* Integration Server. The Mitem*View* Integration Server is a Java (or Visual Basic) compatible application that provides the following core integration services:

- Efficient Message Processing
- Event-driven Business Logic
- System State Management
- Multi Session Management

Mitem*View* is used to create composite applications from legacy system assets. It also integrates with middleware services, application and Web services and vendor-specific interfaces, like TIBCO. By adopting a non-invasive approach to integration, Mitem*View* serves as a *Universal Adapter*™ able to interface with any existing or future system regardless of the underlying operating system, programming language, database or communications protocol.

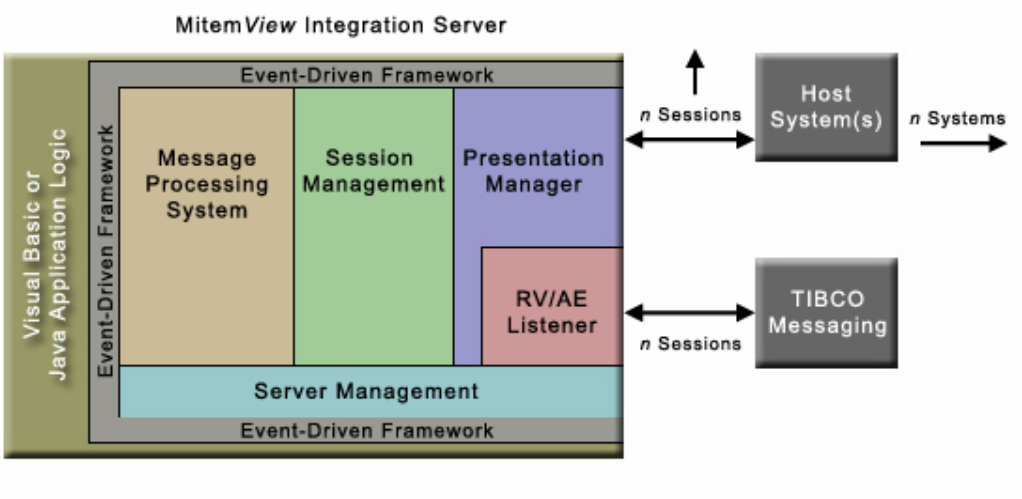


Figure 4: Inside the Mitem*View* Integration Server

The MITEM Tools are a family of productivity assistants that accelerate the application development process, coordinate operational resources and provide control points for managing complex scenarios. Following a unified model, this software suite provides the tools necessary to create coordinate and control a Mitem*View* deployment. The suite includes everything from a wizard-driven application generator to low-level debugging and diagnostic tools.

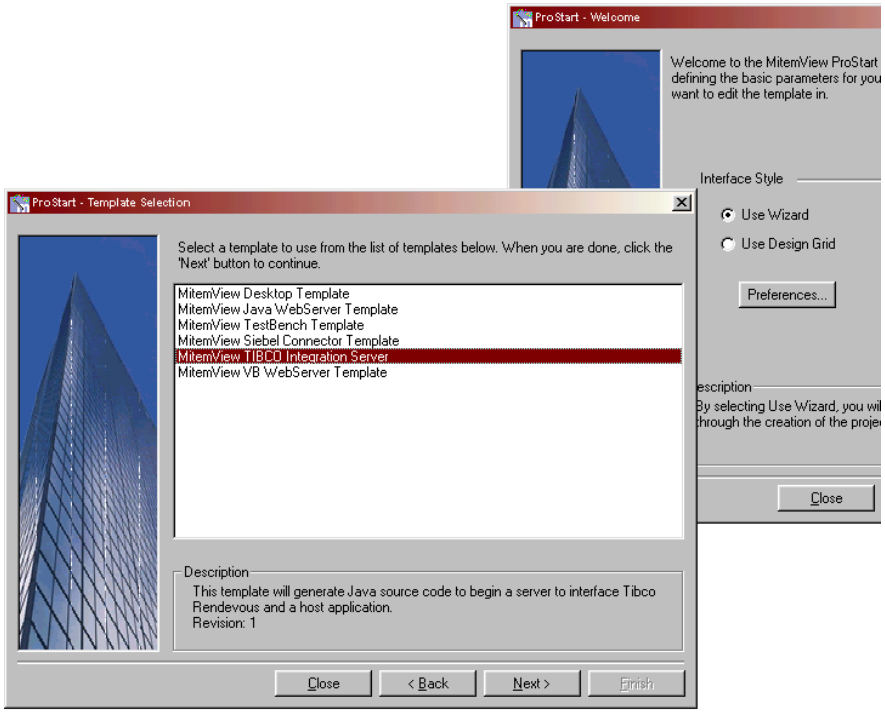


Figure 5. Using ProStart to initiate a Mitem*View* TIBCO Integration Server

The following sub-sections explore Mitem*View*'s core integration services in more detail.

4.2 Efficient Message Processing

In our world all forms of communication depend on the reliable exchange of messages between parties. This is true of human and data communications. As human message processors, we "format" messages according to a known structure. In this way, as the sender of the message, we expect the receiving party to understand the "rules" of the message structure we have chosen. For example, the rules of the English language are comprised of syntax – the grammatical arrangement of words – and semantics, or meaning, of those words. With knowledge of these rules the receiving party is able to interpret the message that has been sent.

In data communications we also define message structures and give each party, in this case a computer system or application, the rules to interpret the message. However, just understanding the rules is not sufficient to guarantee the accurate and efficient transmission of messages.

Consider that:

- During a sequence of messages one or more messages can be "lost"
- A message can be corrupted with or without notice to either party
- A message can be interpreted incorrectly if not fully received
- Excessive "noise" can require the same message to be re-sent many times

The primary benefit of implementing a message processing system to communicate between distributed applications is that well-defined or formal messages reduce the complexity of the receiving application logic. The net result is clean and simple logic that deals only with how to respond to the message received. This, in turn, often allows an immediate response to the sender which, in many cases, can occur in parallel with further processing of the message data. During any "conversation" each application must take turns in being the sender and the receiver, thus the inherent efficiency of a message processing system is enjoyed by both applications.

Without a message processing system the receiving application logic becomes very complex. This is because applications must then contain conditional logic that analyzes the in-bound data streams and determines what, if any, message has arrived and what action should be taken. Such hand-crafted applications are far more code intensive, and are, by definition, very brittle. Each application has the potential conditions and behavior of the other hard-coded in it, so changes in one application can easily break the other.

To implement a message processing system for communicating between distributed applications one must establish the following set of basic services:

- Message Delivery

  The physical transportation of a message from sender to receiver

- Message Parsing

  The encoding, decoding and addressing of each message

- Peer Synchronization

  Message Analysis

- *What message did I get?*

  Method Invocation

- *What should I do now?*

Mitem*View* contains the only message processing system that provides a complete set of services that function equally well across a range of different interfaces – from terminal data streams to formal messaging protocols.

4.3 Event-driven Business Logic

Software developers have become accustomed to communication libraries, which serve as functional enhancements to their development environment. These libraries are accessed via external function calls. In this model the application logic must always call the appropriate function when an action is required. Rather than acting as a service provider to an application, in the way that Mitem*View* does, these libraries demand further procedural coding by the developer.

It is generally understood in software engineering circles that to move easily from procedural coding to a fully event-driven environment one must use a Framework approach. A Framework executes in parallel with your application logic and provides services through a very high level Application Programming Interface (API). As a service provider the Framework controls "when" code is executed, leaving developers to focus on

writing only the code they need – the message handlers. In contrast, when using a communication library one must write the code to control the "when" as well as the "what". We can summarize by stating that Frameworks, compared with libraries:

- Reduce the total development effort
- Execute more efficiently
- Produce code that is easier to maintain

The Mitem*View* Framework defines the location of the message handlers in an application. For example, in a Visual Basic environment Mitem*View* sends events to controls on a form. In a Java environment, Mitem*View* invoke methods on a Java class that is designed to receive messages. The Mitem*View* Language API understands the appropriate calling interface for each development language.

The characteristics of the Mitem*View* Framework can be summarized as follows:

- Defines a common application architecture for Mitem*View* applications
- Integration control logic is isolated from application GUI and business logic
- Application logic is in control, except when handlers are invoked by Mitem*View*
- The Mitem*View* interface immediately returns control to the application logic
- Minimizes the impact of non-determinism to produce a linear development curve

4.4 System State Management

Regardless of the system by which messages are processed, when two or more applications are co-operating with each other over a network they will encounter discovered states.

Discovered states refer to conditions discovered during execution of deployed applications which the application developer did not plan for in the original design. To end-users a discovered state will usually manifest itself as an application error, hang or crash. It is not possible for application developers to know in advance every circumstance that could occur during program execution. Contributory factors include no such state being recorded in the original system documentation, or the state only occurring 'once in a blue moon' when a unique set of conditions are true. With the awareness that discovered states will always occur between distributed applications we can test how MITEM's message processing system copes.

The message processing system can, as described earlier, eliminate conditional communications logic from the receiving applications. Such applications merely contain the message handlers (logical units of code) that respond appropriately to the in-bound message. Once a new state has been recognized as a previously unseen message, it can either be mapped to an existing message handler or an entirely new message handler can be defined. Because message handlers are distinct and isolated code units, introducing a new message handler does not involve altering or rewriting any existing application code. Thus the risk of destabilizing an already functioning application is removed.

This simple approach is in stark contrast to other methods of processing messages that require conditional logic in the receiving application. In such cases the conditional

procedural logic would always be affected by a discovered state. Consequentially, discovered states cause such environments to incur very high costs of development and maintenance.

4.5 Multi Session Management

Mitem*View*'s session management infrastructure incorporates the concept of a *device* and a *pool of devices.* A device represents the network connection, protocol, and presentation-specific attributes of a communications session. For example, a device (also referred to as a session) may carry IBM 3270 data over SNA or a Digital VAX data stream over TCP/IP. The device methodology provides a consistent way for Mitem*View*'s integration services to handle each message and state.

A server-based implementation model often requires a *pool* of devices that are dedicated to support the Mitem*View* Integration Server. Take, for example, a self service Web application that allows customers to query and update account information (that works in real-time with a legacy host system). There may be hundreds of thousands of customers that use this application, but only a small percentage of those need access to the legacy 'Customer Information System' at any one time.

Mitem*View* allows for a *pool of devices* to be allocated to the integration server. This feature minimizes the number of active connections that need to be reserved for the requesting application (e.g. the Web self-service application). Depending on the requestor's requirements a small number of sessions (say 20) may be designated as 'anonymous' devices to the remote application. These 20 pooled devices are then managed on-demand by Mitem*View*.

The Mitem*View* device pool is a collection of devices that are primed to be in a 'ready' state to start a transaction. When a request comes in the integration server picks a device from the pool. That request, coupled with a host device from the pool, create a Mitem*View team*. The team is formed so that Mitem*View* can reply to the correct requestor upon completion of the request. This logical representation of a series of resources as a team frees the developer from the low-level coding that would otherwise be required to manage persistent resources. When that request is completed, the team is destroyed, and the device, which is no longer in use, is returned to the device pool so that it may be used again by another request.

5. Mitem*View* and Terminal Data Streams

We have established that all distributed applications exchange messages and that a message processing system is needed to simplify the communications code. We have also established that the non-invasive integration method considerably eases implementation and in some cases is the only available option. So the question becomes: how can we use a message processing system to communicate between applications that have no apparent formal message structure, without substantially changing those systems to "formalize" them?

The answer is to "externally" impose a formal message structure on the target system. For example, the terminal data stream is the only ubiquitous, non-invasive application interface provided by every legacy application. But a terminal presentation has no apparent formal message structure. However, on closer examination it is possible to see the contents of a terminal presentation as "messages" to be processed by a human message processor. Using a combination of instinct and training, a human message processor (the end-user of the application) performs the same rudimentary tasks of deciding where the message begins and ends, what the message actually means, and what to do with the data.

These tasks are directly analogous to those performed by a computer message processor and thus a formal message structure can be imposed.

Human message processors use framing clues to process the messages in a terminal presentation. An example of a framing clue is the layout and format of the screen, which is typically padded with lots of "white space" to help the human message processor to recognize the meaningful information. This padding can be more than 50% of the total data packet (one can consider an 80 column x 24 row terminal presentation as a 1,920 byte packet of data), and serves no other useful purpose. Another example is field labels along-side every data value, which are necessary for human message processors to parse the data structure, but are not required by a computer message processor. All of this "noise" is redundant data to a computer message processor, and becomes an obstacle to imposing a formal message structure.

To effectively impose a formal message structure one must find a way to eliminate the "noise" and expose the messages. The deceptively simple answer is to write the application code that filters out the "noise" from the messages. Hand-crafting this type of code may result in an application that appears to work, however the complexity of the code will again increase the length of the development project and will consume CPU cycles at runtime. In addition, by embedding the code to eliminate the "noise" in the application logic, we introduce the risk that changes in the "noise" – such as the format of a terminal screen changing – will break the application. The solution is to use a message processing system that eliminates the "noise" and identifies the messages independently from the application logic.

We have established that message parsing is an essential messaging service. Message parsing is the process of encoding and decoding data for transportation between systems. Mitem*View* has Presentation modules that encode and decode data according to the standard specification for a given protocol.

For example, MITEM's 3270 module is built to IBM's specification for the model 3278 and 3279 terminals. This specification controls how the data is displayed, hence the familiar 3270 terminal screen. The open and flexible design of Mitem*View* allows it to process

every terminal data stream in existence. MITEM has already created Presentation modules for every common terminal presentation and MITEM has delivered custom modules in less than 30 days.

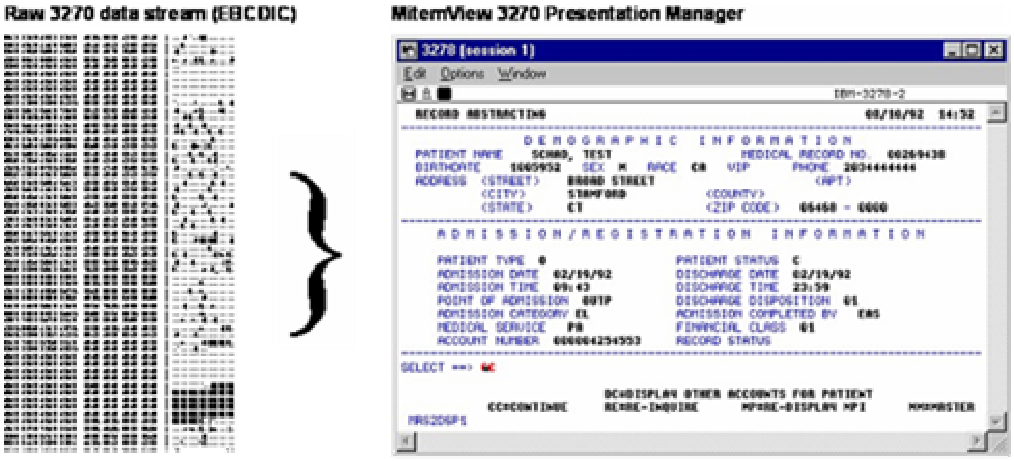


Figure 6: Decoding a raw data stream and presenting the data in a Presentation window for processing

Achieving connectivity between different systems is only part of the integration problem. The operational factors will truly determine whether a new system is accepted and used in earnest by the intended user population. Mature products like Mitem*View* inherit operational characteristics that only come from constant use in many mission-critical applications. The following sections speak to the five most critical characteristics that combine to create a true enterprise-class integration server: performance; reliability; scalability; security and system management.

5.1 Performance

Mitem*View* is designed for performance-intensive data processing environments. For instance, in the early nineties Mitem*View* was tested against the IBM TPF (Transaction Processing Facility) and passed with flying colors. The TPF system was developed by IBM for the special high-volume, high-availability needs of the airline and other reservation systems. Outside of the specialized real-time data feed systems, as found in stock exchanges and so forth, Mitem*View* has been tested in every heavy-duty data processing environment.

Mitem*View*'s system performance is gated by the remote system response time, not the other way around. Internal tests have shown that Mitem*View* adds approximately 2 milliseconds overhead to standard I/O processing. We describe this as near-zero-overhead. The fastest host system that MITEM has found since implementing our server product was installed at a bank's core processing center. Against this system Mitem*View* processed 40 screens of data per second per session, with bursts off 55 screens per second. In other words, with 10 concurrent host sessions, Mitem*View* would process 400 screens of data per second. With 100 concurrent host sessions Mitem*View* would process 4,000 screens of data per second, and so on.

Many of our call-center implementations are configured to provide each agent with multiple concurrent host sessions. This configuration provides maximum performance and workflow flexibility. For example, at one of MITEM's power utility customers each agent maintains 4 concurrent host sessions. This configuration allowed the customer to design a single *dream screen* that displays information from 34 IBM CICS transactions. This look-

up process completes in less than 4 seconds and gives each agent sufficient data to answer the majority of incoming calls without additional processing.

5.2 Reliability

Mitem*View* is designed for 24x7x365 operation in complex data processing environments. Other legacy integration products appear to be designed only for "success" and offer crude error management and recovery capabilities – if any at all. MITEM understands that down-time not only costs money and lost productivity, but also adversely affects customer service and perception.

The Mitem*View* Framework was designed to presume system failure. MITEM's unique message processing system, coupled to an event-driven framework, ensures that system state information is always maintained and that the systems are synchronized. There is patented pattern-recognition technology that sits at the core of Mitem*View*. In addition, all meta-data is stored externally from the business and integration control logic in a central resource file. This file contains the resource definitions of all messages, data structures and so forth that are likely to be affected by changes in the system environment. By externalizing meta-data in this way, it is much easier to build Mitem*View* applications the right way the first time and maintain them.

To affect the most rapid problem-to-solution turnaround, Mitem*View* includes an innovative SnapShot feature. SnapShot is a Mitem*View* function that can be executed when an unexpected host state is detected. On receipt of a SnapShot command, Mitem*View* will capture the state of the application at the exact point of failure. This information is packaged in a Mitem*View* SnapShot file and automatically routed, without any further user intervention, to the Help Desk or Development Group for analysis. A SnapShot file cannot be viewed or edited by the end-user of the application, thus ensuring the integrity of the data.

A SnapShot file contains the information from the originating system needed to completely recreate the environment for that session on the developer's PC. Once the SnapShot is loaded by the developer, all the usual development and debugging tools operate as they would have if the developer were actually connected to the host system when the SnapShot was recorded. Any necessary changes to the meta-data are saved to the application's resource file. This SnapShot system has been proven to enable problem-to-solution turnaround times of 15 minutes or less.

Scalability

In *n*-tier architectures, the Mitem*View* server(s) should be managed like any other application or web server on the network. MITEM has tested Mitem*View* with numerous third-party system management products, from software solutions like Microsoft's Application Center to hardware solutions like Nortel's Alteon range of switches. MITEM's Professional Services Group (PSG) has extensive experience in working with customers to develop resilient and manageable implementations.

5.4 Security

MITEM's mainframe customers, in particular, are concerned about maintaining very high levels of security. When Mitem*View* is integrating with a mainframe system it is, in effect, behaving like a human operator sitting at a terminal - albeit considerably faster and less error prone! Mitem*View* can perform every action that an equivalently authorized human operator could perform. Each Mitem*View* session must still present valid credentials to

gain access to the system and each Mitem*View* interaction can be recorded in audit and log files. Security credentials can be requested from the end-user for every mainframe interaction, or they can be stored for the life of a session. Equally, for less sensitive transactions like inquiries, some customers choose to create generic credentials that are specific to the Mitem*View* server and shared amongst a pool of sessions. Mitem*View* does not presume any particular security model and gives each customer the flexibility to strictly follow their corporate security standards. Mitem*View* has been implemented with all major mainframe security systems including RACF, TopSecret and ACF2.

5.5 System Management

Mitem*View* inherits the network and system management features of the underlying operating system. Mitem*View* can be configured to expose internal system information that, in turn, can be presented through a standard system management console.

Mitem*View* has been tested with industry-standard software distribution and asset management systems like Microsoft's Systems Management Server and Computer Associates Unicenter. In the future the MITEM Host Adapter will be integrated with TIBCO Hawk functionality.

6. Mitem*View* and Formal Messaging

MITEM natively supports formal messaging protocols by fully implementing and insulating vendor or community APIs inside pre-built Mitem*View* adapters. These advanced adapters offer multiple advantages over the typical low-level API implementations that are found in other integration products:

- Consistent development experience across all protocols
- Substantially less programming
- Graphical interface for message processing
- Faster message processing at runtime
- More robust error handling & recovery
- Externalization of host dependent meta-data
- Easier to maintain integration links
- Connects with external repositories

MITEM's Universal Transaction Manager allows different message and document types (e.g. XML) to be transmitted over different protocols and transports (e.g. TIB Rendezvous or Active Enterprise) in a normalized way. By insulating the developer from working with low-level APIs the learning curve is reduced dramatically and message formats and protocols can be changed in the future without requiring a re-write of the integration control and business logic.

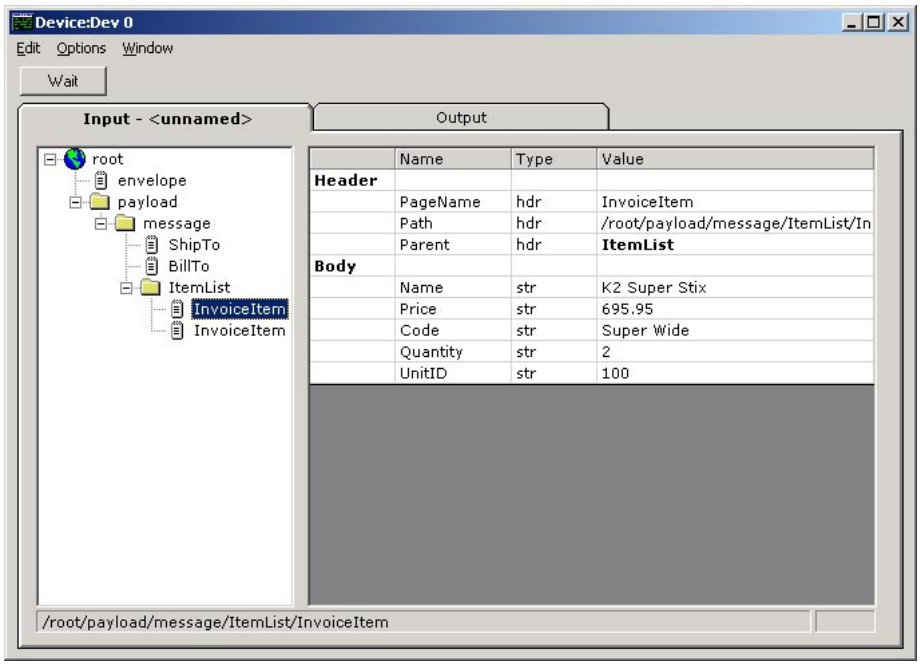


Figure 7: Mitem*View* processing a TIBCO message

Other integration servers that look similar to Mitem*View* on the surface have only crude, if any, support for messaging protocols. Rather than normalizing the development experience they require discrete knowledge of vendor or community APIs. This low-level approach greatly increases the cost and complexity of interfacing between multiple systems by forcing the developer to learn the programming model for every protocol. In contrast, a developer that knows how to use Mitem*View* to interface with one system automatically has all the requisite knowledge and skills to interface with any other.

A further defining characteristic of the Mitem*View* framework is the real-time processing of messages across multiple in-bound and outbound connections. This real-time, asynchronous processing enables maximum flexibility in *n*-tier server architectures and produces superior end-to-end system performance. Mitem*View* performance levels meet or exceed the near real-time performance of other messaging systems, while eliminating the complex conditional programming required for coordinating multiple transactions. Furthermore, multiple host transactions can be combined, or marshaled, into a single message thereby reducing the total number of transactions that need to be developed and managed.

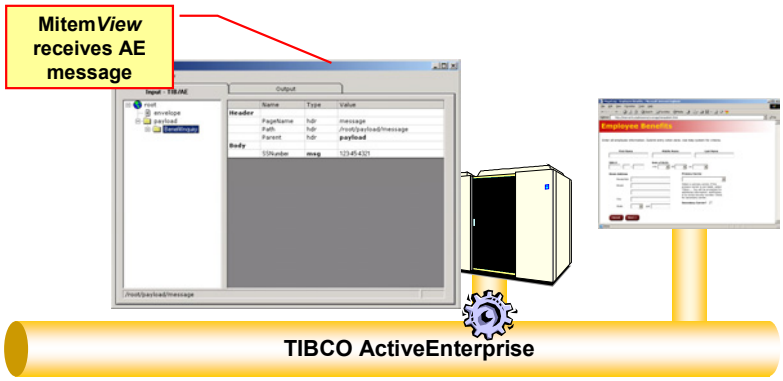
7. Integration with TIBCO Rendezvous and Active Enterprise

In addition to the functions of the Mitem*View* Integration Server, which manages the interoperability to the host system(s), the Host Adapter integrates directly with TIBCO Rendezvous and Active Enterprise:

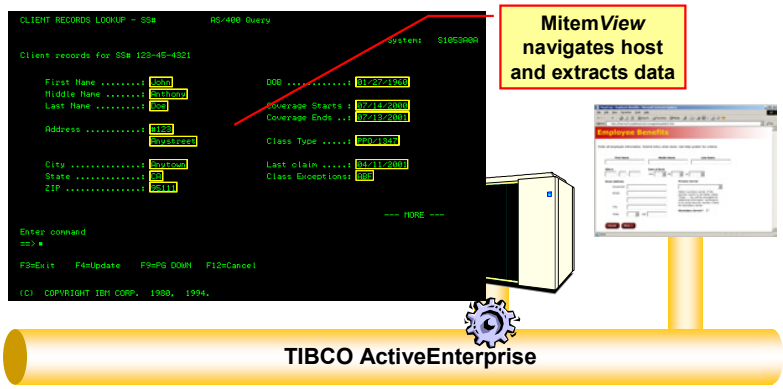
1. **ProStart:** The MITEM Host Adapter can be rapidly deployed. A custom Mitem*View* adapter is generated from a template using ProStart, a tool from MITEM's family of productivity assistants. This tool collects all relevant parameters via a point-and-click Wizard-driven interface. The developer adds custom integration control and business logic to the framework using Mitem*View*'s configuration tools.

2. **TIBCO Message Transformation**: The Host Adapter also brokers the applicable messages to and from Rendezvous or Active Enterprise. The Host Adapter supports TIB/Rendezvous in Certified and Reliable modes and the ActiveEnterprise wire format.

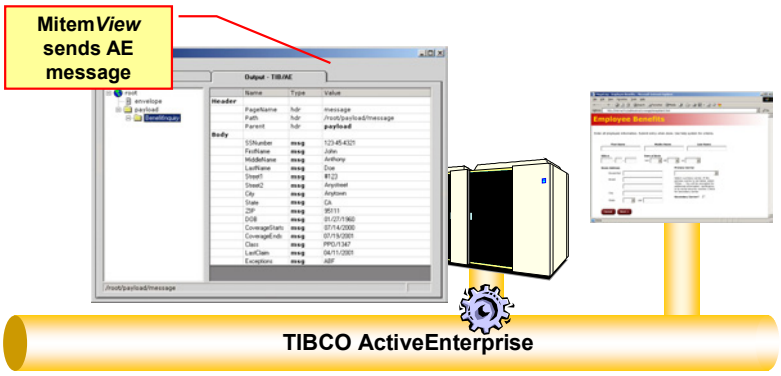
A Mitem*View* Rendezvous listener, as a component of the Host Adapter, listens for an applicable subject from Rendezvous. The Host Adapter recognizes it, and the Mitem*View* application parses it and puts all requisite data into containers in the Java application.



STEP 1: Mitem*View* processes an in-bound request from the TIB



STEP 2: Mitem*View* navigates the appropriate host system and extracts (or commits) the required data items



STEP 3: Mitem*View* generates the out-bound message and posts it to the TIB

Upon completing of the parsing, Mitem*View* initiates an event to kick off the host interaction. Upon completion of the host interaction, the Mitem*View* application creates a responding Rendezvous message, and sends it off to the TIB.

MITEM uses the TIBCO Adapter SDK to maintain common adapter functionality, such as sending and receiving information, configuration, and management and monitoring. The Adapter SDK also enables interoperability with other TIBCO ActiveEnterprise products, including TIBCO Hawk for event-driven adapter management, TIBCO Repository for storage of configuration and metadata information, and TIBCO MessageBroker for data transformation and analysis.

8. Technical Specifications

System Interfaces:

IBM mainframe, IBM AS/400, Digital VAX, UNIX (any flavor), Unisys mainframe, Bull mainframe, ICL mainframe, Tandem mainframe, Data General, Hewlett-Packard and others.

Presentation Modules:

IBM 3270, IBM 5250, DEC VT420 (includes VT100, VT102, VT220), Tandem 6530, Bull VIP family, Data General 470, Unisys UTS-60, Unisys T-27, HP series, TTY and HTTP/MIME.

Formal Messaging:

TIB/Rendezvous, TIB/ActiveEnterprise, IBM WebSphere MQ (previously MQSeries), XML over HTTP, HL7 (Health Level 7) and ANSI X.12.

Network Protocols:

TCP/IP, TN3270E, Novell SAA, Microsoft SNA, DLC (802.2), LUA, TNVIP and Async.

Client Platforms:

Windows 95, Windows 98, Windows XP, Windows NT 4.x. and Windows 2000.

Server Platforms:

Windows NT Server 4.x, Windows 2000 Server, Sun Solaris (due 2002), Linux (due 2002)

Development Languages:

Microsoft Visual Basic and Java (JDK 1.1 and above).