Middleware makes mainframe applications more productive at APL

Michael Woods
American President Lines
and
Allen Cull
Principal, API

Management introduction

American President Lines (APL) is a US$3billion global company providing container transportation by ship, rail and truck. In this interview Michael Woods of APL and Allen Cull, formerly of APL and now a principal at consulting firm API, discuss one method they used to integrate distributed and legacy systems in a mission critical application environment at APL.

The interview recounts how the Company was fortuitous enough to discover, over the span of a few years:

- *the full extent of the technical problem*
- *the solution*
- *additional opportunities presented by the chosen solution.*

Mr. Woods and Mr. Cull describe one specific technique and product (MITEMView) to integrate mainframe and distributed applications in a client/server structure where the mainframe applications must, for one reason or another, be accessed through terminal data streams. They then explore what the solution gave APL.

How it began; how HLLAPI is not all

Our understanding of the problems surrounding legacy integration through terminal data streams evolved over several years. In that time we really learnt what was required in order to solve it.

The use of middleware at American President Lines, and MITEMview in particular, goes back several years to when Allen was assigned to a group called End User Development. For several years this was a separate organization, outside the MIS department, focused exclusively on the end user environment.

The business requirement — the event — that started everything took place when Allen was in an operations group at a warehouse supporting business processes that were then using a variety of IBM 3270 based mainframe applications plus some manual steps. A manager thought he saw an opportunity to automate parts of certain processes, and assigned one of our programmers to build a custom PC application which would:

- collect certain mainframe-based information through the 3270 interface
- automate specific processes (that is, drive mainframe programs, including running transactions, through the 3270 interface).

This PC application did what today we call 'screen scraping' — reading and writing raw 3270 data streams from and to mainframe applications. The program used an interface provided by IBM called the 'High Level Language Application Programming Interface' (HLLAPI) through a product sold by Attachmate.

We called this PC application 'Host Access Interface' or HAI. HAI was just a simple program, written by End User Development without MIS involvement and with, perhaps, 20 commands. It was not robust enough for a production environment but it took a lot of pressure off everyone.

Thus, in general terms, what launched us on this path was:

- someone seeing a need to automate some internal clerical processes
- a programmer figuring out how to make a PC able to interact with a mainframe application.

Overall, the key reason why APL started with this type of middleware is that we, in the End User Development group, were not allowed to use the development resources of MIS, nor could we obtain the extracts of mainframe data that we needed. We had, therefore, to use existing facilities — either production reports or ad-hoc reporting capabilities that were available on our systems. We were forced to gather data by accessing reports in electronic form or screen displays or both, and merge these together.

Introducing the user and Mac dimensions

Later, when Allen joined the MIS organization, someone who had seen HAI said 'we need something that will work on the Macintoshes that will do the same thing that HAI does' (APL owned a lot of Macintosh computers then). Someone in a preliminary planning research group knew of — and suggested — MITEMview which supported the Macintosh. Soon APL was a Beta test customer for Version 1 of MITEMview.

Back then (1989), MITEMview supported a HyperCard-equipped Macintosh (it now supports PC server and Web server versions). The early product did most of the difficult work of capturing mainframe application screen sequences and driving (automatically running) mainframe 3270 programs.

One attraction of this was that a programmer would only have to 'show' MITEMview the mainframe application by exercising it under the development environment. The programmer would then specify data movement in and out of 3270 fields, as well as write scripts. The product took care of the really hard work; in addition it offered run-time efficiencies.

Our earliest efforts automated what users could do directly themselves, because at that time we did not feel we could build a production-quality application. We needed the safety of knowing that — in the worst case, if our program completely died — users could always go back to their original procedures.

Eventually, people forgot how to use their old procedures. In addition, the newly hired never learned them in the first place. Also, since the new method was more efficient, staff levels declined — which was good — but this meant that reliability became ever more important.

For example, problems could become really disruptive. Our programs might break because someone made changes to the:

- underlying mainframe applications
- screens
- network configuration.

Our difficulty was that we were not always staffed sufficiently to keep abreast of these changes. Nor were we in MIS (at that time). As a result, we came to know the pitfalls as well as the benefits of trying to drive mainframe applications from desktop computers. That said, we used MITEMview because it:

- delivered what was needed, especially compared to living-with our own earlier home-grown solution(s)
- offered significant value compared to our home-grown efforts.

Automating work processes

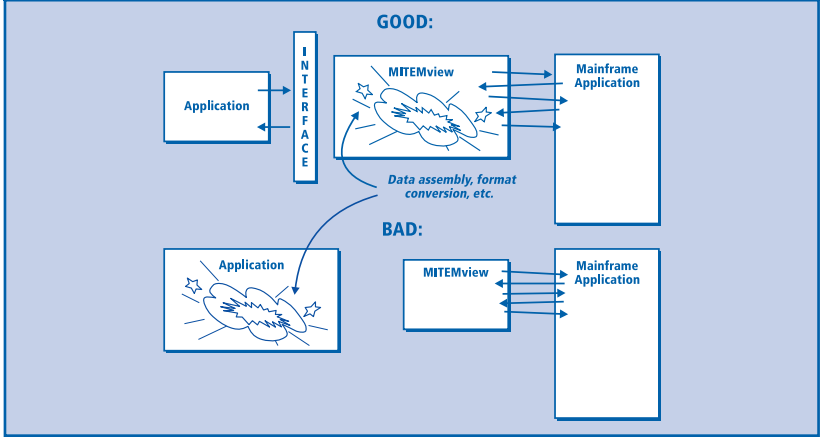
As described above, we were tasked with automating or simplifying work processes. As designers we would visit people where they worked and ask 'what do you do?'. They would respond with descriptions like:

- 'I order such-and-such a report
- when I receive it, I read it
- I then do some additional lookups on the computer for some supplemental information
- I key it all back into another system
- finally, I run another report to make sure that it all worked.'

Any time they were able to articulate their work process like this we could script the sequence of steps, using MITEMview to interact with the existing mainframe applications. For example, through MITEMview, a Macintosh might:

- drive IBM's Time-Sharing Option (TSO) function to start the correct job to produce the appropriate report
- use TSO a second time to look into the resulting mainframe print queue file to read the report electronically and find the necessary data
- start a transaction program to obtain supplemental information from the mainframe's databases
- initiate other transaction programs to put the processed information back into the mainframe applications.

Figure 4.1: Automate



As you can see, this involves not one but several different mainframe applications (Figure 4.1). It requires not merely obtaining information but the use of several different programs. We needed middleware that was mainframe aware.

Improving the user interface

Automating processes was not the only reason we connected desktops to the mainframe. By the time

Michael had joined the MIS group we were also using it to produce, on the Macintosh, graphical interfaces to mainframe applications.

Since we had the Macintosh's GUI at our disposal we were able to simplify the presentation of mainframe screens — sometimes gathering the data from several mainframe screens and pulling it together onto one graphical window to make it simpler for the user to understand.

Thus, the application work we did can be divided into two categories:

- simplifying or improving transaction processing
- automating work processes that were well defined.

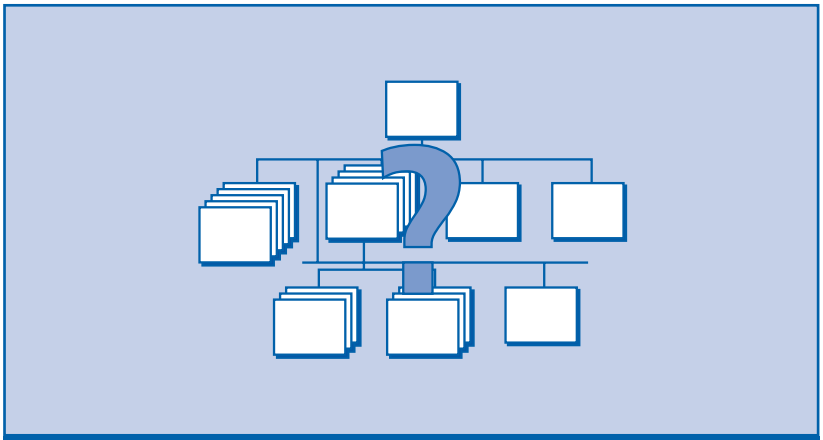


Figure 4.2: Navigating can be hazardous

Contrasting HLLAPI and MITEMview

MITEMview was not the only middleware which worked. APL dabbled in various other HLLAPI-based middleware. However, the MITEM approach (compared to the HLLAPI one) was far more robust. With MITEMview we had a lot fewer problems dealing with the reality of frequently changing screen formats and other situations than we had with those other products.

In fact, using a HLLAPI product was almost as difficult as trying to write something like MITEMview itself within your own application. With a HLLAPI based approach we had to script all of the X-Y coordinates — the exact physical location — of each field in each screen. This made the work to scrape the screen labor intensive and as such virtually unmaintainable.

In contrast, using MITEMview, it was a simple graphical exercise to capture and define a pattern for a mainframe screen and map its fields to a data structure. There was no coding involved. This allowed us not only to define a screen very quickly but also to change quickly whenever necessary.

Another big issue, between HLLAPI based products and MITEMview, is that MITEMview controls the screen presentation. That is, it delivers the screen you want to you. With HLLAPI the application must spend a large amount of effort trying to understand the condition (Figure 4.2) of the mainframe screen:

- is it still waiting for the mainframe to respond?
- has all the data arrived?
- is it on the screen we expected it to be on, or is it on an error screen (and if so what screen is it on, and what do we do now to go back to where we want to be)?

Thus, with HLLAPI a huge amount of code is devoted to determining whether you are in the desired spot within the right mainframe dialog. It was the responsibility of the developer to handle all these conditions and combinations which — in our experience — is impractical.

In contrast, with MITEMview one does not have to deal with any of that. MITEMview runs the mainframe dialog as specified. It indicates to the program 'I am here (on such and such a screen)'. All the programmer has to do is write against these events.

In fact, one of us once had an opportunity to make a direct comparison between the two approaches. Someone had written a component using Attachmate which amounted to 44 pages of program code. We rewrote that component in a few days using MITEMview. The result was 11 pages long and most of that was comments and header boxes.

That same project, using HLLAPI, even reached one point where they (the developers) said 'we just

cannot do this with what we have, in these changing conditions.' It was the booking 'routes' screens — mainframe screens which could be in either of two different formats, and start and end at any time.

In HLLAPI, these were especially difficult dialogs with which to deal. Yet MITEMview made the problem tractable for APL. We just created maps against both formats, figured out within our application which format was in use and continued.

Middleware solving business problems

Over the years, we at APL found other opportunities and compelling business reasons to exploit our ability to use the MITEMview approach to middleware. Many of these were mission critical, including:

- simplifying employee training via a GUI
- making information available
- middleware reliability requirements
- enabling a hands-off policy.

Simplifying employee training via a GUI

This related to customer service and training of customer service representatives (CSRs). A few years ago APL wished to consolidate customer service into a couple of locations. It was expected that many of our existing CSRs — who were then scattered around numerous locations — would choose not to relocate but would leave the company instead. This meant we would have many newly hired CSRs who would need to be trained quickly.

We felt that creating a friendly new system would be much better than trying to make the new CSRs learn 50 old mainframe screens and their relationship with each other as well as with the customers' requirements. Our task was to simplify the process of making bookings, to speed up the learning process for the new CSRs.

Of course, the easier we made it for the CSRs, the better job they could do for our customers. Indeed, in the process of building the client dialogs for the new system, the development team captured the knowledge held by the existing customer service representatives, which represented a gathering of understanding obtained over some 15 years of service.

Of course, building easy-to-use dialogs is easily done in the client/server world. But the question was how were we going to match up:

- the client/server world and its advantages
- the requirement for the power, security, and robustness of the mainframe environment.

This was one dimension to the emergence of the requirement for connecting the PC world (as clients) to existing mainframe applications. MITEMview made a major contribution here — because we could use existing applications but with utterly new 'front ends' which were appropriate to the new CSR function.

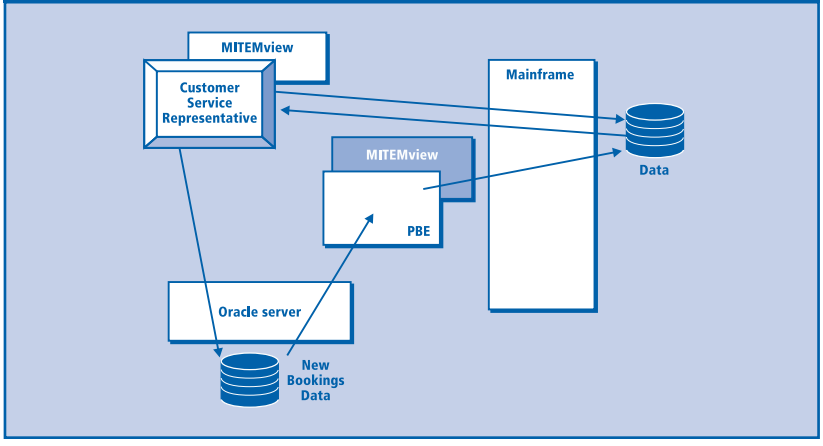
Making information available

A second objective was to add value to information by making it more accessible to more people. The plan was to move a set of useful information off the mainframe and onto a relational database where it could be more readily used by various groups.

The issue was how to do this, when the data in question was part of a highly integrated, mission critical mainframe application which could neither be ripped apart nor moved. The application in this instance was the booking application.

At some point our corporate management also became interested in moving to a client/server environment with Oracle, principally to:

Figure 4.3: Bookings



- give non-MIS users a relatively easy-to-use environment
- allow such users more readily to work with significant data.

However, the transition path was bound to be difficult because the mainframe application and its databases were integrated into every thing else the company did. We could not just move the booking application elements out. We could not segregate the booking process onto the Oracle platform entirely — because it needed so much of what was on our huge legacy system.

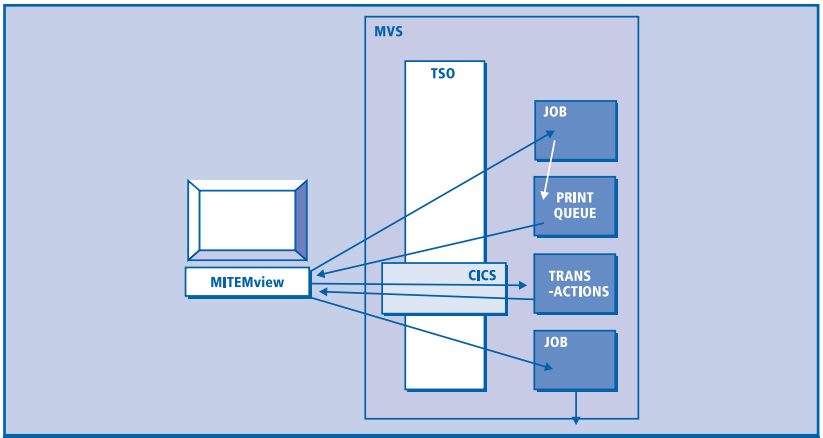


Figure 4.4: The object dimension

To add to the challenge, while we had efficient and reliable TCP/IP networks in North America, we had remote applications all over the world where we were lucky to have any telecommunications at all. For these reasons it was decided that the mainframe still had to hold the core data, the data of record.

In addition to all of this, because of past problems with mainframes, the Company also wanted to be able to continue with bookings even if the mainframe went down. To achieve this capability, we needed to create an environment in which:

- the basic information that the CSRs required in order to take a booking was stored in the Oracle server
- company-wide information (such as schedules, routing, current vessel schedule and remaining capacity of ships) was on the mainframe.

Thus a system was designed as follows:

- the Customer Service Representative retrieved information as required from a combination of current mainframe information (obtained via MITEMview) plus data from the local Oracle system
- the booking record was assembled as the order was taken from the customer — and staged on the Oracle system
- an uploading process — running in background on the Oracle server — detected new confirmed bookings and moved the information back up to the mainframe (using MITEMview).

This completed the circle and made the new booking data available for everyone and every application across the company (Figure 4.3). When this background process, which we called the Perpetual Booking Engine, detected a new completed booking it retrieved the information from the Oracle database and handed it to a MITEMview script which then:

- accessed the appropriate mainframe screens
- populated the input fields correctly with the data
- submitted them to the mainframe, updating the databases through existing mainframe transactions.

At this point the booking was visible throughout the entire company. If there was an exception condition (if the posting did not work) this result was captured on the Oracle side for handling. MITEMview provided the enabling middleware to make this happen.

Middleware reliability requirements

This booking 'environment' was, and is, clearly a mission critical application. It depends on:

- correct operation by the scripts (our MITEMview applications)
- MITEMview itself
- the mainframe application behind.

One element that helped us was that the booking application on the Oracle server performed extensive data validation. By design, the data is very clean by the time it reaches the standalone upload program. It is unlikely to provoke an error response from the mainframe application.

Secondly, MITEMview is itself — in our experience — highly reliable. To throw out a number, we probably have almost a 99% success rate in uploading without having to deal with errors. Our experience is that the only reason the upload might fail is because of changes in routing or pricing data on the mainframe that occur in the short interval, between when mainframe data is retrieved to support the booking and the actual upload (the return trip) happens.

We do not see any MITEMview or mainframe failures. Virtually all the errors that we see are due to bad data. MITEMview itself is extremely robust and has never — in either of our recollections — been the cause of a failed upload. If the mainframe is running (assuming we code the upload application correctly) it works. MITEMview is not one of the weak links in the APL system — which is a positive characteristic for any middleware to possess.

Enabling a 'hands-off' policy with mainframe code

In general, we use MITEMview when we need to solve application problems:

- that require access to mainframe systems
- where the Company does not want to modify those systems to accomplish that.

With MITEMview we are able to be creative. We can obtain what we need from the mainframe just the way it is today — even pulling disparate data from multiple mainframe screens and merging all this together. MITEMview enables APL's policy of protecting good mainframe code, by providing a means of accessing legacy applications that is at the same time:

- reliable
- capable (in terms of being able efficiently to access multiple legacy applications accurately and concurrently)
- truly feasible (overcoming the programming and maintenance problems usually associated with screen scraping).

In summary, the data and applications are out there on the mainframe. The initial information may not be formatted or presented the way we want it. But with MITEMview we can:

- access the data and function
- reformat and recombine it
- present it to a new application as if it were a single mainframe transaction.

Lessons learned

One lesson we learned many times is that mainframe applications and screen formats constantly change. We learned this early on — the hard way. If you cannot easily maintain the interface to the mainframe the inevitable changes create problems.

Another lesson we have learned — an approach that we know we can benefit from if we decide to take it — is that we can look at MITEMview as if it were a database management system. From the point of view of an application working with data — for instance, a PowerBuilder application — there should be no need for that application to be concerned with how or where MITEMview is retrieving or updating the data. We should be able to think of MITEMview as something that can retrieve an object for an application. That is, the application:

- requests data from an interface and the underlying logic — MITEMview and our scripts
- goes and does what it needs to do and then brings the requested result back to the application (Figure 4.4).

Such an application is easier to maintain if we separate functions properly — letting MITEMview take care of the entire task of accessing the data and transforming it into a form that is meaningful to the application, rather than having MITEMview simply retrieve raw data and hand it to the application to sort out.

What we observe is that we can introduce good object oriented techniques by encapsulating the behavior in a MITEMview object. Indeed, the more we compartmentalize and standardize what we let MITEMview do, the more useful it can be. The benefit is that we have code which is more reusable and portable as well as more understandable and maintainable.

Indeed, MITEMview today is a highly structured, highly object oriented product. What we have found is that the more object oriented techniques we apply to our development, the more we benefit from using MITEMview.

From this viewpoint, the approach is soundly based on good object theory. We do not think there will be a limiting factor in our design if we choose to introduce object oriented techniques in our applications — even though the genesis for our experience was originally screen scraping, HLLAPI and mainframe access. MIITEMview, for middleware, has brought us a long way.

Management conclusion

Like many early middleware users, APL set about to solve a particular type of problem before standard middleware products were available. Consequently, it learned to understand its problems thoroughly enough to:

- *recognize the value of a suitable product when one such did emerge*
- *exploit the opportunities that the product brought within reach.*

What APL encountered was threefold:

- *the essence of legacy integration is driving applications, not merely 'accessing data'*
- *often not one, but several, mainframe applications had to be accessed — a major challenge for the average programmer*
- *working with mainframe screens entails myriad details and constant change — the work can be both fussy and seem almost unmaintainable.*

A key issue at APL was, therefore, feasibility. The so-called 'simpler' solutions, while adequate for limited cases, were found to be inadequate for ones with typical complexity, scope or volume. Another issue was architectural.

As Mr. Woods and Mr. Cull describe, the APL approach to mainframe access was to create a set of reusable objects out of mainframe access routines built via some middleware. In addressing this, an immediate technical problem was solved which has emerged as a business asset for the future.