TPF scripting – Thoughts and experiences

By Cees Mul, DATALEX - Amsterdam

Introduction

When I started my career (if you can call it that), TPF was still called ACP, and we were even then largely isolated from the rest of the IT world, as we still are today. It was a bit of an oddball environment and hard to discuss with IT friends who worked on other kinds of systems. Deep inside I was convinced that we were up to something good, but it was hard to imagine that such a niche system, maintained by a bunch of freaks, could compete against contemporary IT developments, such as CICS, IMS, etc. I have remained in that niche for more than 20 years. Actually, it is quite a comfortable little niche, with lots of soul mates that usually agree with you, and with whom one can share thoughts about the latest fashions in IT that bear little or no relationship to IT as we know it, being TPF of course.

Wherever I refer to the plural 'we' to indicate a step in the evolution of the scripting ideas, I mean the project group. Many people have contributed their part. All of the development work so far was done under the KLM flag.

I think that one of the main assets of TPF is its huge existing application portfolio. In order to create web access to the existing business functions, most TPF shops set up a 'shell' that connects web applications to the traditional TPF screens. This technique (screen scraping) had been used before in PC based products, called QIK res, Easy res, Corda accel, etc. We all agreed that this was not a very elegant solution, causing poor performance, software distribution problems, nasty dependencies, etc. There was no alternative, however, apart from rewriting the applications on TPF, which was not considered a serious option. Gradually, more people became aware of the fact that remote applications should not be dependent on entry formats. With that idea in mind, server middleware was developed on things like NT to shield the de-central applications from the TPF formats. See figure 1. Various companies made it their business to build and provide such services. The middleware, residing on a middle tier server, typically takes care of pooling of LNIATA's, and of combining existing TPF entries into new business oriented functions. The web application does not 'hard code' any TPF commands, but calls the middleware for the TPF specific functionality.

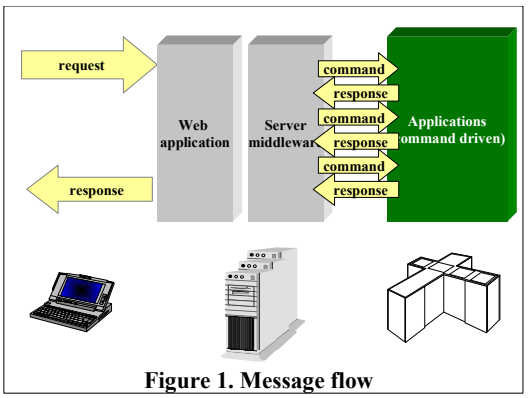

Figure 1. Message flow

This causes a message flow as depicted in figure 1. Business functions that could be described as components are activated by the web applications. This solution works, but it has its limitations, as noted previously. From a performance point of view, it is diabolical. For example, I have seen traces where a single availability request (for three days) expanded into more than 60 requests flowing back and forth between the server and TPF. An availability display contains a lot of irrelevant data, but it is completely read in by the server anyway. This method (using multiple Move Down commands generated by the server middleware) can cause the overall response time to last a full minute. To me this is completely unacceptable in a real-time transaction processing environment.

The first steps

Hence, it was generally accepted that things should be done differently. What are the options? Again, rewrite applications functions on TPF, or build specific screen independent applications. We decided to go for a third approach: push the screen dependencies into TPF. I started playing with this idea some years ago, even before the web presented itself as a potential huge market place. But even then, it was obvious that users needed to be provided with interfaces better than the green screens. People were gradually adapting to windows type interfaces, clicking on boxes to make things happen instead of typing strange strings of characters representing

commands such as '0KL641Y23JUNAMSJFKNN1', although this example makes perfect sense to me. But then again, even assembler makes perfect sense to me.

In order to achieve this data format independency we started, back in 1994 with an OS/2 front end and using SNA server as an intermediary between TPF and the windows dominated office environment. This worked, but OS/2 was overtaken by NT, and we ported the coding to NT. We used this migration to hide the TPF dependencies in DCOM objects. This allowed NT applications to interface with TPF without them being aware of terminal id's and AAA's. Even entry formats were hidden from the NT applications by using scripts stored in TPF. We called the concept 'TACO' (TPF Application Connector).

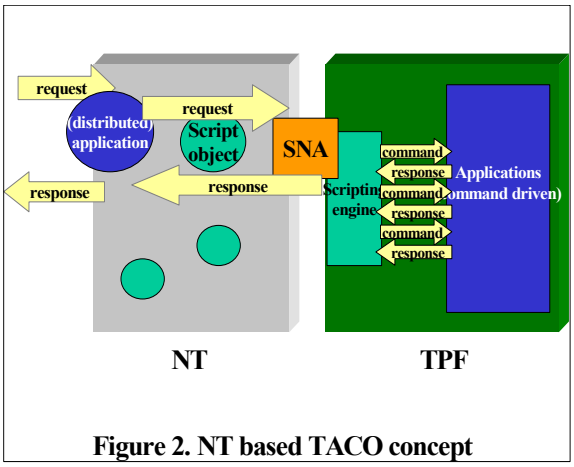

Figure 2. NT based TACO concept

NT serves as a gateway between SNA and TCP/IP, but also opens up the TPF applications for server applications. On the left side we can picture web-based applications, presented by HTTP, WAP, or whatever other TCP/IP protocol we want. However, the same channel can be used for specific client implementations, such as self-service kiosks. The NT Taco component worked in close cooperation with the TPF business function side. In practice, the concept is used on a fairly large scale within KLM. In figure 2, a passenger check-in would mean that, once the passenger data has been gathered by the client PC, the application in the middle tier calls a component on TPF, called e.g. 'check in'. The application provides TPF with the required set of parameters, such as name, date, flight number, possibly the frequent flyer number, etc. The component 'check in' (residing in TPF) contains the required entries, the variables are filled in with the data as provided by the client, and the script is run. The TPF application responses are intercepted by the scripting engine and only when the entire script sequence has been performed, is a message returned to NT. This will be either a successful return with the data that is required, or an error (in both cases in the form of a parameter list). During processing of the script, it is possible to collate data from the TPF outputs, and collect it in a shopping basket, or re-use the data to initiate subsequent entries. From a performance point of view, this sequence looks like one transaction to the calling application. We have seen over a hundred messages a second being processed by a script without jeopardizing TPF performance. Think of all the time-consuming communications exchanges between tiers that have been eliminated!

The main disadvantage of the TACO approach is the dependency on NT and DCOM, and the proprietary scripting syntax that we set up to allow testing and loading of the scripts on to TPF.

Further standardisation

To further standardise the scripting interfaces, we decided to use SOAP[1] (Simple Object Access Protocol), based on XML, to interface between applications and the scripting engine. SOAP can be transported as data, and we started using it across MQ series, using TCP/IP all the way. This flavour is now used in the KLM production environment. Internally, the SOAP call is translated into a script call, transparent for the calling application.

The current scripting implementation has evolved over a period of 7 years from a very basic idea. Now consideration is being given to transforming it into a commercial product. This would mean that parts of the package need to be rewritten in order to make them independent from KLM specific middleware layers and interfaces. In addition, it may be desirable to

[1] SOAP is developed in the internet environment, and is based on a number of open and published standards. It enables Remote Procedure Calls (in SOAP format) between platforms through standard communications protocols, such as HTTP and MQ series.
See: http://www.w3.org/TR/SOAP/

create an alternative for the scripting syntax as it now exists. We believe that a commercially viable scripting engine is a serious option.

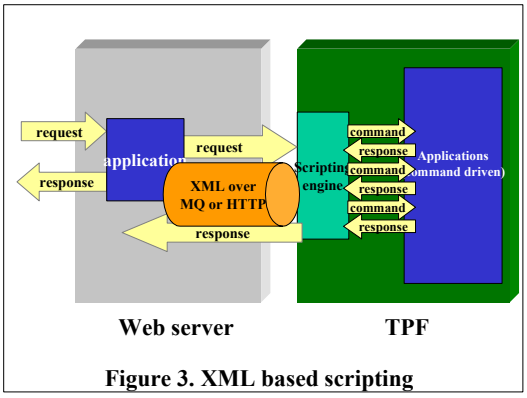

Figure 3. XML based scripting

This picture indicates that no additional boxes are required to obtain connectivity with TPF. The web server could be NT, Unix or any other platform that speaks its languages. The TPF functionality comes as a set of library functions with published interfaces.

Future plans

The previous chapters contain a very simplified view of some of the things we have done in the past. I have not mentioned:

- How to create, store and maintain the scripts on TPF.
- How we know which city code a AAA needs when a request comes from another server, involving a pool AAA.
- How exactly the internals of the scripting engine are set up. I.E. How are the TPF commands processed in the proper sequence.
- In what form (syntax) are the scripts stored on TPF.
- The security database associated with all of this. This was introduced because the time that an LNIATA told you where a terminal was located is over. Security, authorisations, etc. should be based on a user id. This is what the security package does.

In order to make it look simple from the outside, a lot of logic needs to be present under the bonnet.

In the ideal world, it should be transparent to a web platform whether it connects to TPF or another kind of server. In our target situation a platform that supports SOAP and HTTP(S) can access TPF functionality without TPF specific provisions. All airline specifics are shielded from the web server behind SOAP components.

It does mean, however, that web developers have to cooperate with TPF developers. Instead of considering the strange TPF inputs and outputs as being set in concrete, the two groups must try to cooperate and, depending on the nature of a component, decide which platform would be most suitable. Even if TPF is chosen, there is still the choice between writing a script, using an existing script, or writing a new, message oriented application. It may not sound like rocket science, but believe me, it sometimes seems like a religious battle to make these two worlds connect. I am convinced that only a holistic approach offers the maximum benefits, combining the best of both worlds (TPF: large scale transaction processing, applications close to their database, reliable, etc. versus WEB: multitude of protocols, presentation services, common interfaces, etc.). It takes management vision to obtain the best of both worlds.

Even if nothing else happens, having been involved with these concepts and having seen the capability evolve has been great fun. Life can be very boring if you do not have some fun.