A Layman's Guide to TPF

Authors	:	Cees Mul & Bruce Taylor
Original Version	:	1.0, July 2002
Last Update	:	7 August 2002
Issued by	:	Cees Mul
Reviewed by	:	Roland Moor

1. Introduction

This booklet aims to help non-technical professionals in the travel industry, such as sales and marketing people, project managers, ICT managers and senior consultants, who do not have a TPF background, but need to have a basic understanding of TPF for the role they play in their organisations or projects.

There are an ever-increasing number of projects where systems integration, involving the marrying of old and new technologies to achieve a business solution, plays a key role. In the travel industry, TPF systems are often the main background application and database servers. Thus, TPF will play an essential role in integrated solutions for a large number of marketing requirements. It is assumed that the readers of this booklet have experience in the ICT industry, but have never been exposed to TPF (and TPF'ers). The intention is to give the reader an impression of the main characteristics of TPF in the first few chapters, with more in-depth descriptions in the subsequent chapters.

We have attempted to be as unbiased as possible. However, it is difficult to ignore 20-30 years of experience in this very special environment. An unbiased view on any subject does not exist, which is an opinion in itself. Everybody forms his of her own version of 'the truth' according to his or her own experiences. We shall try stick to the facts, but we are TPF'ers to the marrow. TPF is one of the oldest surviving software platforms on the face of the earth: it has been operational since 1968. In those 35 years TPF has weathered all storms and endured all fashion shifts, from the decentralisation of the 1980's through the Client-Server rage of the 1990's into the current Internet age.

The list at the back of this document is intended to help the reader in the acronym jungle, an unsavoury feature in any technology subject, whether young technology or old. Unfortunately, the use of a number of unique TPF acronyms is unavoidable in a document such as this. Over the years acronyms have come and gone and have always been most prolific in the ICT industry (was called IT, IS and many other things in the past). The creation of new acronyms, often simply older concepts in new guises and masquerading as magic, continues apace. There are a number of TPF specific phrases that are explained in the list, but we did not bother to include generally known ICT phrases such as TPC/IP, Unix and so on.

2. What is TPF?

2.1. But first: What is an operating system?

TPF is a *standalone* operating system. An operating system is a piece of software that is capable of making a suitable computer configuration run and perform useful work. This software understands and communicates directly with the hardware components of the computer system, such as processors, disk drives, tape units, consoles, communications adaptors, etc. and tells them what to do. I.E. it understands the hardware architecture of the machine and knows how to direct its activities through sets of commands, which are peculiar to that architecture. The hardware architecture that TPF understands is that of the IBM mainframes, which have evolved from the System/360 of the 1960's through to today's Z-series.

On the other side, the operating system offers the run-time environment for the applications that perform the actual business functions. The latter is called the *API* (Application Programming Interface), which is a set of commands that applications may issue to request specific services from the operating system, such as: "give me some memory space", "read this database record for me", "file this item to the log tape", "send this answer to a user", "I'm finished, please put me to bed", "I have my knickers in a twist, please kill me and clean up", and so on. A given API consists of many dozens or even hundreds of commands with a syntax unique to that API. Like every operating system, TPF provides systems functions and an API that allows the applications to do everything needed to perform business functions. It is therefore comparable with IBM's Z-series OS (previously known as OS/390, before that as MVS, MFT, OS/360), VM, UNIX, VAX/VMS, UNISYS' OS2000 and Microsoft's Windows and NT.

2.2. The TPF operating system

Most older, mainframe oriented operating systems were designed primarily for the processing of *batched work* in large volumes. The processing of online, *real-time* transactions came later as an afterthought. The PC originating operating systems were designed as online, real-time, transaction processors, but were oriented towards a single-user, serialised, low volume processing capability. UNIX is, in our view, a hybrid between these two extremes, though it seems more like a mainframe operating system than anything else.

TPF is, we believe, unique in that the fundamental focus of the operating system design was to offer a transaction-processing platform with the emphasis on (in arbitrary sequence) performance, scalability and availability (*PSA*)[1]. Batch processes, a necessary evil even in a TPF environment, were relegated to a low-priority, background activity. Since this design work was done back in the middle of the 1960's, it is a matter of conjecture whether the people involved got it right by accident or by real design insight. That they got it right is indisputable: 35 years of predominance and current capabilities,

[1] Refer to Chapter 5 for more details on 'Performance, Scalability and Availability'. The only other operating system we know whose design was transaction oriented is the ex-Tandem NonStop Kernel, now part of the HP NonStop Enterprise Division. Its availability equals or exceeds TPF's, but its performance and scalability have never been proven to anywhere near the same extent.

in PSA terms, which are still orders of magnitude beyond what any competing system can achieve, all prove that.

Most other operating systems were initially designed for batch-oriented processes, or were designed for use on PC’s. Banks, for instance perform massive nightly application runs to process their accounts and accumulate data collated from the various branches to print settlement slips, address labels, etc. Those other mainframe operating systems also offer transaction-processing facilities, but usually as a separate ‘monitor’ function that is controlled by the main operating system. After decades of tweaking, these monitors are still the only software platforms that come anywhere close to TPF in PSA terms. The PC operating systems, which vendors tried to scale up to handle greater volumes in the multi-user, transaction-processing environment were, to put it diplomatically, a considerable disappointment.

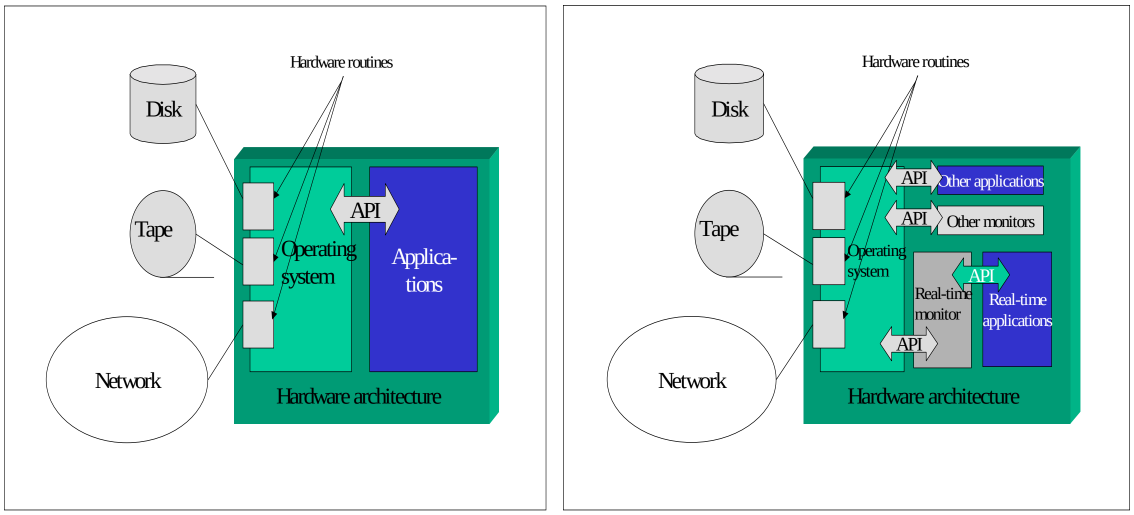

Figure 1: Stand-alone operating system (TPF) and real-time monitor (e.g. CICS, ALCS)

TPF stands for *Transaction Processing Facility*. And that’s exactly what TPF does; it processes transactions in real time. This means that customer requests, whether airline check-in or Internet booking transactions are processed immediately. This characteristic is extremely important for many businesses. A seat in an airplane for instance can only be distributed once. With hundreds of passengers checking in on a flight, even a delay in processing of a few seconds could cause duplicate seat allocations. Because of the nature of the travel industry that spawned TPF, emphasis has always been on performance, combined with a large numbers of users and the requirement to be always on the air.

TPF is an *event driven system*. This means that it sits quietly until a user sends it a command. Only then an instance of an application (for example a reservation) is initiated, and starts doing what that application has to do. In the real world, unfortunately, things are much more complicated than this. For a start, it seldom happens that there is only one user active. Even the smallest of TPF installation supports hundreds of *concurrent users* at any one time; the larger systems, hundreds of thousands. “Concurrent users” is the term used to describe the number of simultaneously connected active users. A typical TPF system may have hundreds of thousands or even millions of potential users, but they will normally not all be logged on simultaneously. Apart from being spread around the globe, and therefore being subject to different time zones and office hours, most users perform their activities in bursts, e.g. when a customer calls, when an airplane departs, or when a flight is rescheduled. If all connected users were to enter a transaction every few seconds, any transaction system would be drowned by thousands of messages. Fortunately, human behaviour is different. A travel agent for instance will switch between a conversation with a customer and the reservations system, causing a limited number of transactions, and thus spreading the load on the system.

The event driven nature of TPF has determined the systems architecture. TPF is controlled by a *task manager*, which schedules the application requests in a very specific sequence. The philosophy behind this multi-processing model is that short chunks of application work are picked up by the central processor, but as soon as a subtask (such as the retrieval of a record from the database which requires an I/O) is requested, control is handed over to a subtask manager, and the central processor picks up the next chunk of work from a list, servicing another application request. This swapping happens so fast that thousands of application instances can be operating in parallel on the one mainframe image. Once a subtask is completed (such as the I/O example) the application is added back to a list and will be processed again when the task manager picks it up. This philosophy only works if the applications loose control on a regular basis: in practice an application stream is never continuously active for more than a few tens of milliseconds and the average is a lot less, usually less than a millisecond.

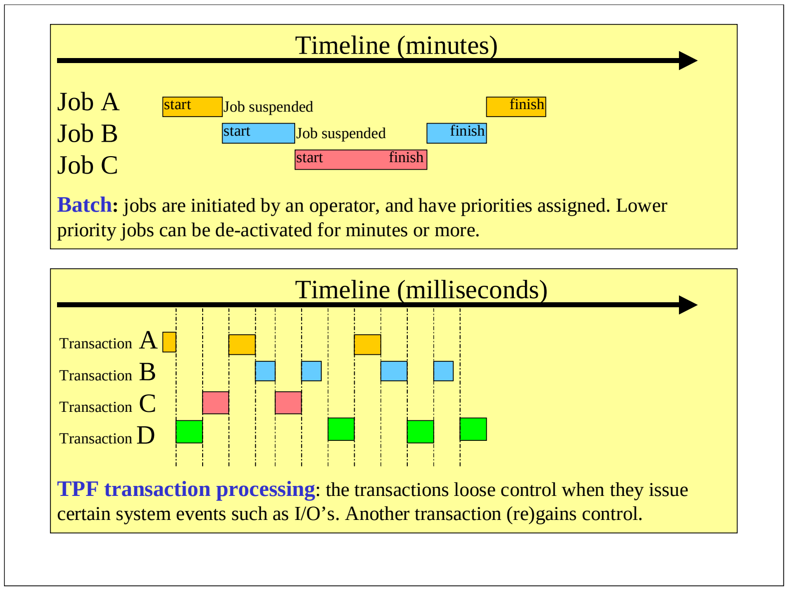
Timeline (minutes)
Job A
Job B
Job C
start
Job suspended
finish
start
Job suspended
finish
start
finish
Batch: jobs are initiated by an operator, and have priorities assigned. Lower priority jobs can be de-activated for minutes or more.
Timeline (milliseconds)
Transaction A
Transaction B
Transaction C
Transaction D
TPF transaction processing: the transactions loose control when they issue certain system events such as I/O's. Another transaction (re)gains control.

Figure 2: Batch versus transaction processing

Another fundamental characteristic of the TPF task manager is the approach that current, partially completed application instances should be progressed as far as possible before starting any new instances. Only when all current instances are awaiting an uncompleted event will the task manager initiate a new instance to service a waiting user request. This is fundamentally different from most, if not all, other operating systems, where complex priorities are associated with user requests and one application instance can pre-empt another in the race for processing cycles. TPF's simple approach maximises the total throughput of a given hardware configuration.

2.3. Batch processes

Even transaction systems need databases: flight schedules, customer data (*PNR*'s) for airlines, inventory for hotel systems, customer account data for banks, etc. Those databases are critical for a business and need regular maintenance runs. For instance, processes are required to purge expired accounts, create a new operational period for a flight schedule, purge past passenger data to archive files, perform consistency checks, expand the database to cope with business growth, etc. This type of routine maintenance activity is handled by batch runs and usually operates at night to avoid unacceptable performance peaks, which might jeopardise the end-user response times.

Apart from application batch processes, a multitude of TPF system support functions exist. For instance, on a regular basis the entire database is scanned and validated by *Recoup*, a TPF systems function that guarantees the integrity of the database structures. All TPF installations have some form of database back-up mechanism that can be used in case of database corruptions. This is either done by the standard TPF tool called *Capture* or, increasingly, by using the ODR (*Online Data Recovery*) facilities of modern storage devices that allow for data transfers without affecting the operating system.

TPF is not shut down for the user world during this type of activity and processing continues normally. TPF can do all of this "on the fly". Only in the case of catastrophic, widespread database corruption would the system have to be brought down to restore a captured copy.

So, despite the fact that TPF is essentially a transaction processing system, a number of very important, batch-like processes do exist, all associated with protecting the integrity of the databases, which are so crucial to the business processes.

2.4. The operating system versus the applications

An operating system is not much use to anyone without meaningful applications. For people without a TPF background, the separation between the operating system TPF and its applications may require some explanation. Phrases such as 'code share', 'direct availability', and many others have no relation to TPF as an operating system. They are application functions. Typical operating system tasks are:

⇒ support the applications through an API
⇒ initiate application instances as a result of a user request
⇒ clean up after an application instance is finished
⇒ recognise failures of application instances and ensure resources are freed up
⇒ control of the systems resources, including error detection and recovery
⇒ activate time initiated processes
⇒ switch tapes, resource paths, etc.
⇒ perform I/O operations to the database, tapes, operator consoles, communications network, etc.
⇒ recognition of and recovery from hardware failures

⇒ cycle up and cycle down processing

In theory, any business application can be written to run under TPF. In practice, TPF's use started in the passenger reservations activities of the airline industry, moved into other travel providers with similar application requirements and then, with a completely different application set, into the finance industry. Except for a few strange exceptions, TPF gained no further acceptance in other industries. For better or for worse, TPF is, and always has been, driven primarily by the travel industry. That occurred because of the overwhelming success of the PARS (*Programmed Airline Reservation System*) application set, which was cloned around the world and is the application base running under TPF used by all travel industry users, past and present. The only competing system, which ever had any impact was, and still is, UNISYS' USAS equivalent of PARS. USAS accounts for about 30% of the reservations system installations worldwide, but probably only for about 10-15% of the transactions processed.

The bulk of the TPF installations in the world process travel related transactions.

	Travel	**Number of installations**
1.	Airlines	32
	GDS's and hosting services	7
	Hotels	2
	Railways	4

	Finance	**Number of installations**
2.	Banks/credit card companies	25

3. **Others**

Two exceptional installations: Japan Travel Bureau (travel as well, but not a PARS derivative application set, we understand) and the New York Police Department's Dispatching System.

The number of TPF installations has decreased gradually for many years. However, the total number of TPF transactions being processed has been, and still is, increasing steadily. The Internet has had a huge impact on the total number of TPF transactions and that influence will continue to grow. The Sabre system, a GDS that also supports CRS hosting facilities tops the bill with a peak average of over 9000 transactions per second in 2001. Peak times for most travel systems are 10:00 to 12:00 and 14:00 to 17:00 during working days. The average TPF installation currently runs daytime peaks of 500 to 600 transactions per second. Outside of the peak periods the systems are far from idle since the travel industry is global. Even in the wee hours of the morning most airline TPF systems will be running up to one-third of the peak load, whereas the GDS's would be running two-thirds or more.

2.5. The operating system versus the hardware architecture

TPF is based on IBM's System/390 architecture, recently renamed to Z-series. This means TPF uses the S/390 instruction set, the addressing scheme, I/O routines, hardware configuration capabilities etc. S/390 is an omnipresent and commonly accepted architecture worldwide, in particular by large companies. Over 95% of the financial and retail institutions in the US have their transaction processing based on mainframe technology. There are very few large companies that do not rely on S/390 systems; the majority however are not TPF, but the more standard Z-series OS. The Z-series architecture is not proprietary, though IBM is now the only one manufacturing it, since both Amdahl/Fujitsu and Hitachi dropped out of that market during 2001. It is definitely the most widespread and reliable platform in the world.

2.6. History

TPF was not invented nor designed in its current shape, though the fundamental transaction-oriented chassis remains basically unchanged. Like many software platforms it has evolved and matured over time.
Back in the 1950's, at the start of the jet age, the US based airlines were having trouble administrating the rapidly increasing numbers of passengers. The first large scale commercially viable reservations system was PARS (Programmed Airlines Reservation System) and it was based on the young IBM S/360 technology, which was a revolution at the time. PARS is the ancestor of the entire TPF travel application family and was developed in the mid-1960's as a joint effort between IBM, Eastern Airlines and Continental Airlines. The first operational version went live at those two carriers early in 1968. Through the remainder of 1968, IBM together with Aer Lingus, Alitalia, BA, KLM and Swissair created IPARS from that US base (the I stands for International). The European based carriers, being more international and inter-continental, had slightly divergent business requirements from the mainly domestic US focus of the American carriers.

PARS and IPARS are application packages. They handle flight schedules, availability figures, passenger data, etc. The operating system under which they ran was originally called ACP (*Airlines Control Program*). Compared to contemporary systems, ACP was very basic; it only supported a limited set of different hardware devices and communication protocols and, conversely, much of this hardware (in particular, terminals and printers) was specifically designed and manufactured for the ACP environment only. Large public networks did not exist, and most ACP installations were forced to control and maintain their own networks. The total number of terminals connected to the system has always been very high right from the early days (starting with thousands and growing to hundreds of thousands now). The network was mainly ALC[2] based for interactive traffic plus low-speed telex links for message switching. In the central mainframe complex, disk drives, tape drives and operator consoles were supported, but that was about it.

ACP was maintained to a large extent by the airlines themselves, as well as officially by IBM, and it was free: in the 1960's and 70's hardware was excruciatingly expensive, but software was in general free. Each user installation added their own enhancements or specific changes and ACP enhancements were happily shared between airlines, so it was really an open source system 'avant la lettre'. This anarchy was great fun, of course, but also had its drawbacks. Different varieties of the same wheels were invented at various locations around the globe and each installation had its own, very knowledgeable, group

[2] Asynchronous Line Control – see list of acronyms

of specialists who fought tooth and nail for their own wheels. The introduction of a new type of IBM DASD (the 3350), for instance, required massive re-engineering of the operating system. All this tinkering kept a lot of talented people very happy, but was not always beneficial for the reliability and continuity of the systems.

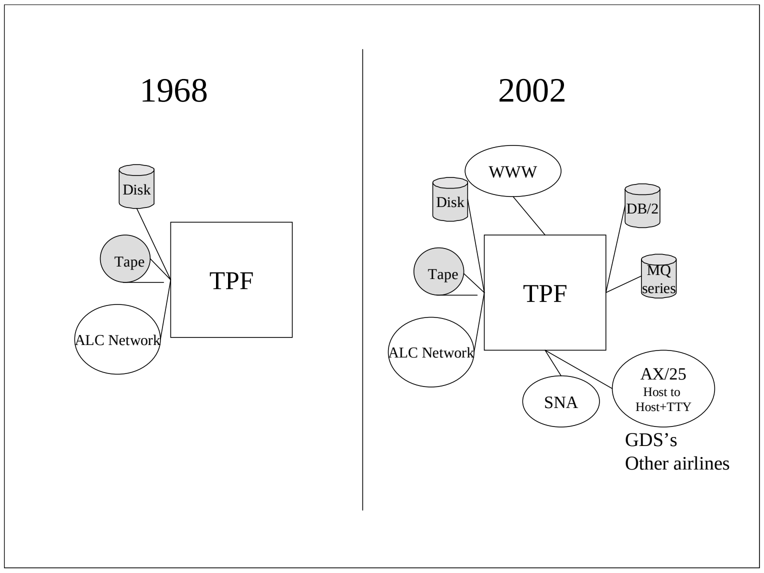

Figure 3: Increase in distribution channels over time

In 1980, IBM came out with a new version of ACP, renamed to TPF, which was a licensed product, i.e. it cost money. From that time on the system gradually matured further, and started resembling a 'normal' operating system. New releases are introduced on a regular basis and the stability and reliability grew to impressive levels. However, IBM still offers customers the possibility to 'do their own thing' to a limited extent by offering user exits. The current version of TPF is 4.1. However, that version has been around for many years now and although the version number has not altered, a lot of enhancements were added over time (IBM distributes a major update tape about twice per year). TPF 4.1 uses the System/390 virtual addressing capabilities, allows TCP/IP connections and XML, and successfully attempts to blend in with major ICT developments. In 2001, TPF based reservations systems processed more than 93% of worldwide airline bookings.

3. Major differences with other platforms

Very often software platform comparison discussions tend to resemble religious warfare. This is common for all comparisons and does not specifically apply to those involving TPF. For example, take the case of the people who defend their Apple computers against the Windows fans. In their opinion, Microsoft’s chairman is the reincarnation of the Prince of Darkness. As usual in the case of religious warfare, they have a point, but they exaggerate. Microsoft knows how to create on-going and un-abating consumer need by producing products, which lock the consumer in, but always have shortcomings, which the next release promises to resolve. A very dubious way to earn money, many might say. The indignant competitors, of course, would never consider such tactics: they are the pure and honest Linux adepts, SUN admirers, Java fans and, yes, there is a group of TPF bigots.

Every system has its own set of characteristics; each one of them is suitable for specific environments and/or specific business requirements. Comparing them in the sense of ‘which one is better’ would be like comparing a lorry with a Formula 1 racing car; they are different because they were designed to perform different tasks. This chapter will list the main differences between TPF and its ‘competitors’. We will try to avoid weighing those differences and merely list them, in order to steer clear of the insoluble religious warfare debate.

3.1. The TPF database

3.1.1. A different animal

One of the major causes for confusion in relation to TPF is its database. The majority of current ICT professionals associate the phrase ‘database’ with either relational databases, such as Oracle or DB2, or with an Object-Oriented database. These all allow a programmer to store and read rows of data or objects and leave the application programmer in blissful ignorance of the exact formats in which the data is physically stored on disk or how the relationships are achieved. The database management system (*DBMS*) shields the applications from the gory details, at a substantial cost in terms of both processing cycles and database I/O operations. The wonderful thing about such databases is that running statistics, management reports and determining relations can all be done with standard calls or queries. Once the data is stored into the database, another application, not necessarily running on the same platform or in the same image, can access the data. There are even ‘bridging’ modules to allow, for example, DB2 database elements to be stored onto an Oracle database and vice versa.

The TPF database is a very different animal and rather difficult to explain to non-TPF’ers. It is not simply, as many describe it, a ‘flat’ file. If anything it can be described as a combination of flat files logically ordered in multiple hierarchical structures. We will try to clarify the concept with an example. All TPF reservation systems have an inventory structure, containing all the airlines’ schedules. For each day that a flight is active, a detailed inventory structure exists. In here the current number of passengers per selling class is maintained. At the same time, passenger name index files are maintained for each active day. These index files only hold the passenger name and a reference to the passenger name record itself. The applications are aware of these database dependencies and of the pointers to other records.

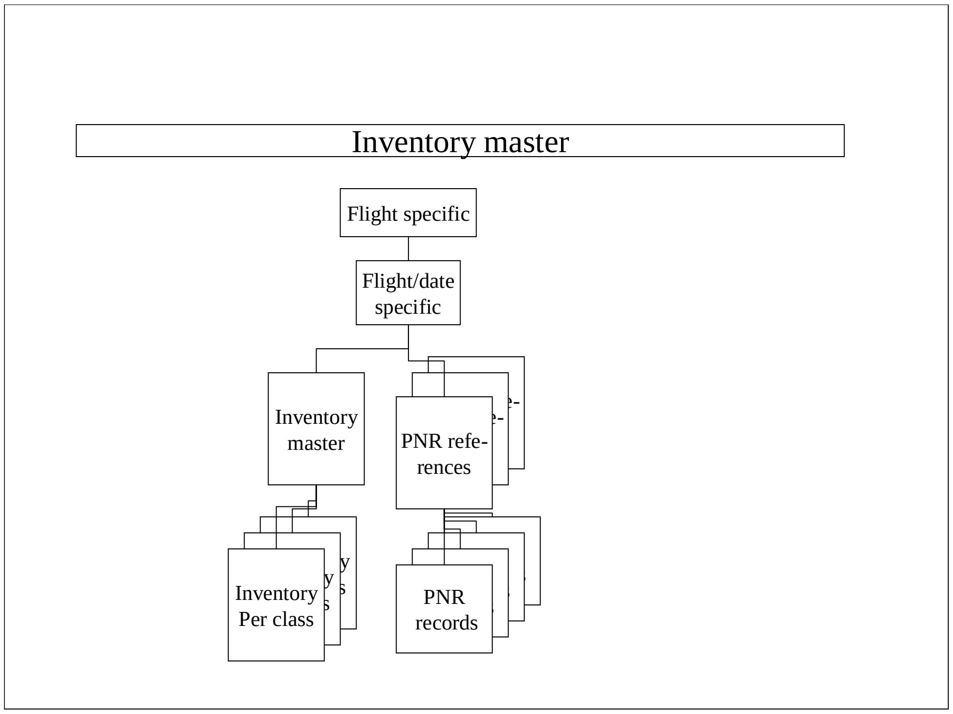

Figure 4: TPF database relations

The data in the records can be a mix of characters and binary data and is only meaningful for a specific application. A PNR display program for instance can only handle PNR records. If presented with an inventory record, it would fail. The reason for this database structure is fundamental to the TPF principle of: PERFORMANCE (the P of PSA).

Every application that updates data in a record is also responsible for updating the related records. E.g. the application must not only update the applicable data but also update any related (index) databases.

Consequently: the TPF database can only be read and/or updated by TPF applications and not by any other kind of application. Any process external to TPF that needs to access the TPF database has to go through a TPF application to do it.

3.1.2. Reads versus writes (a love game?)

Web sites of the Wimbledon tournament or the Olympic games are often quoted as proof of the fact that large scale transaction processing can be achieved in the Internet environment with modern technology servers and no mainframes. Alas, we are again comparing apples and oranges. The sites like the above are examples of predominantly *read-only* databases. The content of the database is static and is only updated from one source. When Andre Agassi wins a set from Tim Henman, a single person enters that fact into the database. There is no risk that other users can or may want to update the same data. Allowing access to many thousands of people is not a problem. Since the data is static, performance can easily be guaranteed by caching.

Now, consider a case where all those thousands of people can actually update the status of the Agassi-Henman match, for instance to voice their opinion about the quality of the match. We are still talking ballgames, but indeed, of a very different kind. The database would be subject to hundreds of changes per second, and in order to provide every user with the most recent status, caching is not an option any more. The updates would have to be processed directly. This scenario would be comparable to what a TPF environment offers, with the difference that most TPF installations run much more complex, and much more diverse, applications than the functionally trivial Wimbledon or Olympics examples. This is where the low-level database lock feature of TPF plays a crucial role and makes a difference between performing and not performing. The database lock mechanism is explained in some detail in the next section.

The Wimbledon tournament application does not require those real-time, multi-channel updates of course, but in other industries it is a requirement to grant customers update-access to a central database that is shared by all. We foresee a growing need for such large scale, real-time applications. We believe that current new technology implementations, similar to the examples above, will not suffice to address the required scale and business needs of the e-commerce world of the future.

3.2. The real-time aspect

This issue is closely related to the previous one. Since the TPF applications are so closely connected to the database, it is possible to ‘lock’ the database at a very detailed level. If an application is, for instance, planning to update a PNR record and the update would not affect any other part of the database (for instance, a passenger’s home address contains an error), then the application would only need to issue a ‘database lock’ on the PNR record itself. The records that are required to locate the specific PNR (which are retrieved by utilities) are retrieved as read only. Multiple instances of such an application can now co-exist without locking the database at a high level. The time it takes an application to perform the PNR update can be expressed in milliseconds. Immediately after filing the updated record to disk, it is unlocked, and the next instance of an application can access and update it. During the lock of a record, all other requests for that record are queued in the system. This queuing of records that request a database lock is a typical example of a task that the operating system performs. TPF application programmers are trained to limit the number and duration of database locks and they possess the tools to do so. Occasionally, a programmer inadvertently violates this basic rule, thus risking severe performance problems, but their peers will rapidly diagnose the problem and fix it.

The very detailed and low-level database locking capability is one of the reasons behind TPF’s excellent performance characteristics.

All TPF activity is performed in real storage. Many operating systems file pieces of main memory to disk files (called paging or swapping) in the case of memory resource limitations. For performance reasons this was never implemented in TPF, since the cost in terms of processor cycles and I/O overhead is considerable. If too many application instances become active in the system and memory becomes constrained, then TPF will start shutting down certain tasks (for instance no new instances will be created, no new input requests will be permitted to come in, etc). TPF solves memory constraints in a way that maximises performance and response time.

3.3. Securing the database

IBM offers a number of crucial functions that are indispensable to support groups. In addition to that, most TPF installations have built support functions to meet their exact requirements. Some examples:

Recoup	Database integrity tool. This software package reads through the entire TPF database, using database descriptors to determine the dependencies between the records and verify the correctness of the structures. It also identifies, verifies and returns free space to the available pool for reuse. Recoup is usually run at night, though not all installations have a big enough maintenance window (i.e. a period where the real time transaction load is low enough) to run it every night.
Capture/restore	Writes copies of all the disk volumes to tape. Those 'capture tapes' are stored in a safe place and are only used in case of dramatic database corruptions.
Logging	Critical database updates are logged to tape. TPF supports two types of tapes: Real Time and common. The real time tapes are continuously mounted. If a volume fills up, the operating system switches to the standby (which is always present). If a record is judged critical enough to be logged, a table definition is adjusted to indicate that logging is required. The real time tapes are primarily for disaster recovery, but are sometimes used in combination with capture/restore to track down and correct database errors and corruptions.
Data collection	Takes measurements of various activities in the system. This tool supports the performance analysts in determining the optimum settings of the various TPF tuning parameters, of which there are a large number. The sort of things that are measured are: average life time of an application instance, I/O completion times, average number of I/O's per transaction, etc. Properly set tuning parameters are key for an efficient and reliable TPF operation; a way to measure the system behaviour in detail is a prerequisite for that.

3.4. Application development

Contrary to common belief, application development in TPF is not essentially different from what it is on other large system platforms. So, why then, is it common knowledge that teaching someone to program in TPF is much more complicated and therefore time-consuming than it is on other platforms? The main reason is not TPF itself, nor the fact that programs can only be written in Assembler or C: it is the PARS/IPARS application set and the fact that thorough, disciplined software development is no longer deemed necessary.

The PARS/IPARS application set has evolved over a period of 35 years and represents hundreds of man-centuries of iterative and cumulative software development by thousands of people. It is a mass of millions of lines of Assembler and C code divided into small inter-dependent segments. In many ways it appears an enormous, complex and inedible bowl of spaghetti. Programming changes to such a 'can of worms' can only be trusted to experts who really understand how it all interrelates and hangs together: to gain such knowledge and insight takes time; lots of time. Whereas in most modern ICT development environments one could expect an intelligent trainee to become fully productive in a few months, in TPF applications it takes 2-3 years.

In the current ICT world, driven by the software development practices of the PC age, where people can create new functionality using Java in hours instead of weeks, the TPF development environment appears hopelessly ancient, complex and time-consuming. People are educated to expect to get, and are told they can achieve, wonders in software development in real time. All the graduates coming out of school think this way, their tutors thought that way, ICT management thinks that way and all consultants preach it as the gospel. However, as usual people are again comparing apples with oranges.

Writing an application is one thing, writing applications that work properly in a large-scale interactive environment, exposed to the rigours and failings of the real world, is something completely different. Writing applications that work around the clock for hundreds or thousands of concurrent users, all needing update access to a highly volatile database, is not only a different ballgame, it is a totally different planet. Application development on TPF is relatively slow: on that we all agree; but that is because it implicitly contains a number of valuable and indispensable characteristics for the target user environment and the critical business processes it supports:

⇒ Through its API's, the TPF programming environment enforces high performance, industrial-strength, multi-threaded code.
⇒ There are very rigid test and acceptance procedures in place, because the continuous availability of the TPF system is a critical asset to a company. As a result of these checks and balances, the quality of the code that eventually makes it to the live system is extremely high. Several testing tools to support the testing exist, like Test Version Load Facility, debugging tools such as Step-by-step Trace (SST) or Visual Age for TPF, Online Dump Facility (OLDF), and dedicated Test Systems for every stage in the development, including a live size acceptance system.
⇒ The TPF operating system shields each application instance from corrupting influences and from interfering with any other instance's resources. Memory leaks, for instance, a common problem in many newer operating systems are well nigh impossible to cause, either advertently or inadvertently.

What is frequently forgotten or ignored by ICT management is that the operating costs of an application over its lifetime are usually a large multiple of its initial development cost. In addition, the lifetime of an application is usually many times longer than the 3 to 5 years originally budgeted. The Y2K panic was a stark example of that. Due to arguments in favour of 'time to market' and other commercial pressures, these facts of life are almost always ignored. In turn, this results in decision-making based on short-term, shortsighted goals and resulting in dramatically escalating ICT operating costs. Furthermore, applications built in a flash and marked as 'ready for the real world', usually collapse in a heap on exposure to the vagaries of that real world. It is the difference between a toy and a tool…

Very often organisations do not even consider TPF when deciding on a development platform for a new application. This is due to the TPF application development myths that are mentioned earlier in this chapter. We do not advocate developing all applications on TPF, but if business requirements for an application match TPF's OLTP characteristics, it is irresponsible not to consider TPF as an option. An application should be developed on the most appropriate platform, depending on its characteristics. An OLTP system like TPF is not the most suitable place to develop presentation services; and a PC is definitely not the best choice for large scale transaction processing. With correct choices, the combination of the two working in unison achieves wonders.

4. The influence of commercial developments

The rapidly changing travel market has primarily determined TPF's evolution. The original PARS and IPARS systems served airlines. The central reservations departments and branch offices worldwide sold seats, controlled the flights, set up the schedules, etc. and only airline personnel had access to the systems. By the 1970's, at the airports, passengers were being checked-in using the TPF based check-in facilities and the first cargo handling applications on TPF appeared.

However, through the late 1970's and first half of the 1980's something had been sneaking up on the industry without many realising its significance: Sabre! Sabre (American Airlines) had been putting ever-increasing numbers of terminals into travel agents in the US: something unheard of in the rest of the world. Apollo (United Airlines) saw the danger and had more recently started doing so as well, but Sabre had a flying head start. What the airlines saw happening was that an agent with a Sabre terminal did two things. Firstly, they booked a lot more American Airlines flights than non-automated agents did, resulting in American Airlines growing at a much greater rate than anyone else in the industry. Secondly, the other flights that such agents did book, incurred a booking fee from the airline to Sabre and Sabre was making money hand over fist. The European airlines, bonded together in the Association of European Airlines (AEA), headquartered in Brussels, suddenly woke up to the fact that at any minute Sabre could start doing the same thing in Europe. If that happened, then the European airlines had no response, traffic on transatlantic routes would shift to American (or United, since they now had the capability in Apollo as well) and the Europeans would be paying booking fees, in ever increasing sums, to the Americans. Nobody liked the sound of that.

To counterattack these developments the European Airlines decided to create their own GDS (Global Distribution System) in the late 1980's. Being Europeans, they were unable to agree on the proposed solution, so eventually two European GDS's emerged: Amadeus and Galileo. Because of the size of those GDS's, both installations chose to use a TPF system. Galileo used United Airline's Apollo system as a base and Amadeus was based on SystemOne, which was the Eastern Airlines system. Air France, Lufthansa, Iberia and SAS were the founding fathers of Amadeus. BA, Aer Lingus, KLM, Olympic, Swissair, Alitalia and TAP were the initiators of Galileo.

Since such a large part of seat sales is done through travel agents (who could decide to become either a Galileo or an Amadeus agent) direct sales links between the GDS's and the participating airlines were required. The result was a significant increase in direct connections. Today we would call that: B2B… The airlines had been exchanging passenger and availability data for years, but the focus shifted from type B (a-synchronous, low-speed) to type A (synchronous, high-speed) messaging between B2B partners. Hence a complex organism of inter-connected reservations systems and GDS's evolved and millions of electronic messages are exchanged daily.

Due to the on-line nature of the reservations systems, it was relatively simple to connect the browser-based applications of the Internet world to the TPF systems. The GDS's were amongst the first to hook up to the Internet. It had always been their business to act as a storefront for the airlines and they quickly adapted to the new channel. Worldspan started a joint venture with Microsoft, called Expedia, now responsible for an enormous, and ever increasing number of Internet sales. If we consider GDS's as 'front ends' for the airlines' systems, the GDS web portals can be considered 'front ends' for the GDS's. Of course GDS's offer much more than just airline seats, but they charge significant fees for their services based on the number of passengers booked through their system. The latest trend for the airlines is to try to skip the intermediate step of the GDS and thus avoid the associated booking fees. In increasing numbers, they are connecting a web server directly to their own reservations system, to be able to offer their flights directly to the consumer, the true power of the Internet.

The widespread acceptance of electronic communications had not gone unnoticed by the developers in the IBM TPF laboratory. It became obvious that the TCP/IP based protocols were going to dominate the communications world, whether they liked it or not. Hence, the most strategic of those protocols were ported to TPF: HTTP(S), MQ series and SMTP (the mail protocol); and are now actively used by TPF installations, enabling further integration with the rest of the world. TPF still processes transactions in such an environment; in fact, it has to process many more. A persistent characteristic of successful e-commerce ventures is that the "look-to-book" (i.e. queries versus sales) ratio goes up and up and up. With traditional distribution channels the look-to-book ratio is usually a few-to-1, but with successful e-commerce sites it easily becomes 50-to-1 and even hundreds-to-1. If the site is successful, then people want to have a look at what it contains, and how it is presented, even though they have no intention whatsoever of actually buying anything on offer. This implies that the transaction volumes the business processor must handle are at least one order of magnitude higher to generate the same volume of sales. A system capable of behaving itself at a few tens of transactions a second does not necessarily do so when forced to cope with many hundreds of transactions per second, even if the hardware is upgraded. Hence, travel companies have to realise and accept that doing business on the web involves consuming a much larger amount of processing capacity. I.E. if doing $1M of business via traditional channels requires the ability to process 10 transactions per second; doing the same amount of business via the web will require the ability to process hundreds of transactions per second. Here TPF is in its element.

The number and diversity of connected access channels has increased significantly. TPF systems now offer data to Customer Relation Management (CRM) databases, e-mail systems, web servers, publish & subscribe services, etc. In the meantime, it still supports the old 'copper wire' protocols, simply because there is still a large amount of traditional airline equipment around, and also everything else invented in the intervening years (BSC, SLC, SNA, X.25, etc.).

5. TPF specific system characteristics

Up till this point we have tried, possibly unsuccessfully to avoid too many technicalities. However, now the technicalities will increase. Hence, the following chapter can be skipped by those readers who do not require a deeper understanding of TPF. The special characteristics of TPF are the consequence of a number of technical choices and are therefore essential for this document. TPF distinguishes itself from all other platforms by its unequalled PSA: the combination of performance, scalability and availability.

5.1. Performance

Commercial real-time transactions need to be fast, even in peak hours when transaction rates may run into thousands per second. We have already mentioned a few explanations for the superb performance characteristics of TPF in Chapters 2 and 3, but here is a partial recap, with a few new items:

⇒ Use of the Z-series architecture.
⇒ Its multiprocessing concept, where the task manager 'time slices' small amounts of application work in very strict sequences and all tuning is focused on achieving a minimum existence time for all application instances in the system.
⇒ Its simple "first in, first out" task scheduling, which avoids the overhead of the complex algorithms required for task priority determination.
⇒ Its avoidance of the overhead of paging or swapping main memory to/from disk files.
⇒ The database structure with its low-level locking capability.
⇒ The spreading of the database across multiple disk volumes. By properly defining the database, excessive queues of applications requests for a particular disk volume can be avoided. Large TPF installations have thousands of (logical) disk drives connected to their TPF systems. By spreading the I/O load across the volumes, I/O rates of many tens of thousands per second can be achieved.
⇒ VFA (Virtual File Access). A main memory buffer that 'caches' the database records read from disk. A subsequent request for such a record results in using the copy, thus saving a disk I/O. Database writes are also stored into VFA and a record is only physically written to disk, unless otherwise specified, when the aging mechanism determines that it needs buffer space freed. In practice, VFA hit ratios (where the system finds the required record in the buffer and thus avoids the physical I/O) of 80-90% are common and a TPF system cannot survive without it.

And last, but not least, the crucial parts of the operating system have been written and rewritten many times over the decades, as new insights into the hardware and software characteristics emerged, to enhance the performance of the whole.

5.2. Scalability

The layman may rightly ask: "If a system works for one, or five or ten people, why won't it serve a few thousand as well? Surely, all you need to do is put in a faster processor and more disks?"

Scalability is a complex issue and it is not just a technical matter, but also a question of economics. How far can a Unix farm be stretched? The theoretical answer is: infinitely, but only as long as the database is static and/or it can be split into unique copies per server. Suppose one server supports 20 users. If the number of users grows, we 'simply' keep adding servers, each with their own database copy. True, but this comes at a cost and only applies to the business functions with the right database usage characteristics. Even if it can be done, the resulting infrastructure becomes ever more complicated, including load balancers, fallback servers, etc. It may work, but hardware and software licence costs will rise at least linearly with the number of users. The complexity of the operations and management, and the footprint of the equipment, also goes up at least linearly with the number of users. Thus there is little or no possibility for economy of scale. These are the scalability issues.

A system that is designed and able to handle a large number of users, plus large and volatile databases, in a single system image, can be expanded without significant extra cost to accommodate growth. For a Unix farm, the added complexity of such expansions results in an increase in the number of support staff and has a negative impact on the overall reliability of the system.

Of course, a system that is very efficient for a large number of users will probably be very expensive if applied to a small number of users. Inversely, a small office application may not work at all for thousands of users, but who cares, since it does wonders for half a dozen people. It is cheap and cheerful.

5.2.1. Typical scalability restraints

What kind of limitations can a system encounter? There may be a few others, but let us round up the usual suspects:

⇒ Storage restraints
This is probably culprit #1. Although storage is available in enormous amounts compared to bygone days, when a Megabyte still meant something, modern software has no problem eating it all up and craving more. This is inherent in the programming languages themselves, in the database handling support and is exacerbated by inefficient storage management techniques, programming errors causing storage to be lost (memory leaks) or simply because a system is not designed to handle large volumes efficiently. Applications in TPF can be restricted with regard to the storage they use, and the operating system itself controls all storage handling. Communications sessions only use limited storage. TPF programmers have been trained to handle their resources with care: for them programming is a craft.

⇒ Queuing delays caused by the database
Earlier in this document, in the section on 'Performance' it was described how the TPF database is spread across multiple volumes to avoid queuing. Most other systems use a standard relational database. The question is, whether a database that is accessed by a high volume transaction system can be a relational database. If only read actions occur, a lot can be achieved by caching, but that is not the nature of a transaction system. Its database is subject to many changes that should immediately be reflected in case another process queries the database.

⇒ Synchronisation overhead
  Scalability is not a matter of 'adding iron'. By adding a number of servers to a complex, the overhead of synchronisation traffic usually grows faster than the ability to do useful work. If there is, for example, only one directory containing user profiles and all servers seek access to that in order to validate incoming messages, then the queuing can get out of hand. In addition, the servers have to inform each other about the states of user sessions, in order to do proper load balancing, and at some point the synchronisation traffic stifles the entire system. In TPF, the Loosely Coupled facility minimises this overhead.

5.2.2. Some figures

During the spring 2002 *TPF User Group* (TUG), a twice-yearly conference, IBM announced a number of new features to further boost the scalability of TPF. TPF now offers support for 32 Central Processor Complexes (CPC's) of 16 processors each in a single image TPF complex: 32 x 16 = 512 processors, yielding in excess of 80,000 MIPS. The maximum TPF database capacity has been expanded by the introduction of FARF6 (8 byte file addresses), which enables TPF databases to grow to a vast 288,000,000 Terabytes.

In the past, the largest TPF installations frequently were in danger of running up against the processing or database capacity limits of TPF. The above measures should keep them in the clear for some years to come.

5.3. Availability

The concept of 'availability' is simple: it is the percentage of time that a system and its applications are available for its users. Unfortunately, this is open to creative interpretation, such as: is it measured as a percentage of planned up time, scheduled up time, actual up time or total time and what is the length of the measurement period? Most vendors will naturally choose the definition that makes their platform look best.

For TPF, it was established and accepted decades ago that availability is always expressed as a percentage of total elapsed time measured in minutes over a whole calendar year (i.e. as a percentage of 525,600 minutes, except in leap years). Most TPF systems have established availability figures of 99,94% and higher (i.e. they are down 315 minutes or less per year); some consistently achieve 99.99% (i.e. they are down only 52 minutes per year). There are TPF installations that have achieved 100% availability for many months in succession. The travel industry has always recognized the importance of extremely high availability for its critical real-time processes. Since many airlines operate on a global scale, there is never an appropriate time to take a system down. TPF is a true representative of 24*365 operations.

Why does a computer system experience outages anyway? Two types of outages can be distinguished:

Planned
Unplanned

5.3.1. Planned outages

Most computer platforms require outages for planned maintenance tasks, such as hardware upgrades, program loads, database extensions, etc. Apart from system related tasks, some applications require outages for things like: database compressions, database re-indexing, making backups, etc.

Very few of these apply to TPF. To minimise planned outages, TPF installations have a mix of procedural and technical tools at their disposal:

⇒ Downtimes for hardware (CPU) upgrades can be avoided by loosely coupling the processors. Loosely Coupled can be compared to the OS/390's sysplex feature. It allows an installation to couple a number of processors sharing a single database, and thus making the Loosely Coupled architecture relatively transparent for the applications. In a properly defined Loosely Coupled complex, individual processors can be unplugged, upgraded, and activated again without any system disturbance.

⇒ Program loads can be done online, using the TPF online load facilities. Programs are loaded in 'sets'. An entire set can be activated by operator commands. After the set is activated, new application instances run through the new program set. This feature is highly integrated with the operating system. It evolved, like most other TPF features from relatively simple, individually developed packages into a standard, very valuable TPF tool. On test systems, it allows each programmer to load and test with his own set of program versions.

⇒ Like most applications, many TPF based applications require maintenance runs. Since it is not deemed acceptable to shut down a TPF application, all maintenance tasks are on-line. This is again a matter of instilling the right 'mindset' in the applications programmers.

5.3.2. Unplanned outages

Unplanned outages are never welcome and are due to programming errors or operating mistakes. A small, 10 person office can more easily be told to take an early lunch due to system problems than companies consisting of ten thousands of people spread around the globe. The economic and public damage caused by a one-hour outage can be enormous. Anyone who has been in a busy international airport when the check-in system is down will know exactly what we are talking about. TPF installations have been familiar with this fact for decennia. As is the case with planned outages, most TPF installations have a combination of procedural and technical measures in place to prevent outages, and if they do occur, to limit the damage and restrict the duration to a minimum:

⇒ As mentioned before, there is a very strict and disciplined regime of software development control: first various prescribed test phases, user acceptance, system integration and only then live load.

⇒ With a high number of changing applications, a watertight version control system is essential. TPF also offers on-line fallback possibilities, again without having to bring the system down.

⇒ Hardware failures have become rare, though they still do occur. The purveyors of special, so-called non-stop, hardware create the impression that their hardware ensures more reliable systems. In reality, software is the cause of the vast majority of unplanned outages. Reliable systems are only created by a mixture of excellent hardware together with all the measures and controls and techniques listed in this chapter.

⇒ The database can be completely duplicated and all systems use that feature. The I/O's are written to two mirror disks, called a prime and a dupe. In the case of failing I/O's, one disk can be taken offline and the system continues. With modern disk drives, even more copies can exist, since the disk subsystem has its own, internal duplication hidden from the software.

⇒ Emergency procedures are taken seriously. Fallback hardware is always present and the fallback procedures are exercised regularly. Well before ITIL existed, most TPF installations (and large mainframe installations in general) complied fully with its best practice standards, unaware of being so far ahead of the rest of the pack.

A small kernel of highly experienced, extremely knowledgeable and highly motivated people is vital for the continuity of any system. People are the most important ingredient in any success story. TPF is no exception to this rule, and a strong, long-lived and globally oriented TPF culture is present in every installation.

6. Epilogue

We hope we have been able to clarify the basic characteristics, strengths and weaknesses of TPF and indicate the reasons for TPF's unique position in the ICT industry and history. Writing this document has been a balancing act between keeping the document concise and the urge to tell everything. The reader who survived all the way to this point, hopefully realises that large scale transaction processing is not just a matter of phrases and acronyms. It requires experience, determination, motivation, knowledge, time (evolution), an open mind and, yes, money to successfully fulfil the mix of opposing requirements, which make large scale transaction processing a unique challenge in the ICT world. Within that environment, TPF is a major player and will remain so for many years to come.

Interesting web sites for people who want to know more about TPF and the airline industry:

www.datalex.nl
http://www-3.ibm.com/software/ts/tpf/index.html
http://www.tpfug.com/
http://www.tpftoday.com

7. List of acronyms

ACP	Airlines Control Program. The former name of TPF.
ALC	Asynchronous Line Control. A 6-bit communications protocol, mainly used by airlines. It is still used, sometimes even wrapped in a TCP/IP envelope (MATIP).
ALCS	TPF's younger brother. A real-time monitor under MVS that supports the TPF API. Used by some of the smaller and medium sized airlines. Philosophy: Supporting a single, commonly accepted operating system (MVS) is simpler and cheaper than supporting both MVS and TPF.
API	Application Programming Interface.
DBMS	Data Base Management System.
I/O	Input/Output request. Applications use and update data that is stored in databases. The processing to access (read or write) the data is referred to as I/O.
IPARS	International Programmed Airline Reservations System.
IPL	Initial Program Load. Tells a computer that it has to restart. A specific hard-wired track on the database contains the initial instructions that kick off the restart process.
Loosely Coupling Facility	Enables Processors to be connected together and function as a single image system. The current limit is to couple 32 Z-series processors, each with up to 16 I-streams. The bigger TPF installations often have a 'communications front-end' that is actually a TPF complex all in itself. Sometimes they have separate sets of processors for different application packages.
MVS	IBM's major mainframe operating system. Formal current name is z/OS. But usually still referred to as MVS. OS/360, MFT, MVS, OS/390 and Z/OS are all different evolutionary stages of the same system. In this document we use the phrase 'MVS'.
ODR	Online Data Recovery. Facilities of modern storage devices that allow for data transfers without affecting the operating system.
OLDF	Online Dump Facility. This is a tool that allows system or operational dumps to be viewed on-line rather than have to wait to process a dump off-line.
PARS	Programmed Airline Reservations System, ancestor of all current TPF based reservations packages.
PNR	Passenger Name Record. A (TPF) file that contains data for a passenger or a group of passengers. One of the most crucial files in the airlines TPF reservations systems.
SST	Step-by-step Trace. SST is a hands-on testing aid, designed to trace the progress of an input message through selected programs, macros and even instructions. SST originates from British Airways.
TPF	Transaction Processing Facility. Read 'a Layman's guide to TPF' to pick up the essentials.

TUG	TPF User Group. The TPF Users Group is a non-profit organization, consisting of member companies that own one or more TPF licenses and maintain primarily for their own use or use by their customers an online, in-house commercial production TPF system or complex of TPF systems. The TPF Users Group meets twice a year and provides an organization whereby professionals may gather with others of like interest for the advancement of the state-of-the-art in the TPF operating system in its various versions and releases. See www.tpfug.com for more information.
VFA	Virtual File Access. A caching mechanism for database records, which maintains main memory copies of highly accessed records and is transparent to the applications.
VPARS	Virtual PARS. VM tool that allows the simulation of multiple operating systems on one machine. VPARS mimics the (TPF) database, and allows individual users to have their own database. Particularly used in test environments.