



# InspireD

---

RFC on Draft SW Architecture Specification - Abstract

# Table of Contents

---

1. INTRODUCTION.....	2
2. EXISTING OPERATING SYSTEM CONCEPTS.....	2
3. CONCLUSIONS .....	4
4. BIBLIOGRAPHY .....	6
5. GLOSSARY .....	7

## 1. Introduction

---

Most smart cards today are very application oriented (such as Banking or GSM cards) even if they are open for adding multi-purpose applications with the JavaCard technology. In any case the smart card conforms to the request response paradigm conforming to the ISO 7816 standards. The function of the Operating System does not really exist since either the dedicated application or the JavaCard Virtual Machine manages the smart card hardware resources that are reduced to the I/O channel and the memory.

Beyond this smart card traditional technology, first the on-chip features, such as processor and memory, grow up considerably; second, new high-speed communication channels appeared, such as USB; third, the connection with peripherals such as display, knobs are possible. The implementation of Internet communication protocols becomes a great opportunity for smart cards to be a network entity, as well as server and as client. Considering this, the hardware resources management of these new smart cards requires more functions than the one present on existing smart cards

However, the *InspireD* project working area is more than just a smart card; it introduces the concept of Trusted Personal Device (TPD) which gathers not only traditional smart cards but also System on smart card, Multi Media Card (MMC) and (USB) tokens.

This document is an abstract of the deliverable "Request for Comment (RFC) of the Operating System Software Architecture Specification" for the Trusted Personal Device (TPD). This much more complete document has been realised in the frame of the *InspireD* project which is an EC-funded initiative (IST-2002-507894).

The document starts with an analysis of the business requirements of the TPD expressed in the D1 deliverable document. The objective is to identify the new hardware resources to satisfy the business requirements and to express this for the TPD hardware architecture working group WP1.3.

The main part of the document concerns the state of the art of operating system functions that are currently used in various devices between the current smart card and embedded system such as Pocket device like Smart-Phone and Personal Device Assistant (PDA). The state-of-the-art covers the Operating System resource management present to offer an abstract machine interface that is a Virtual Machine or Interpreter. The resource management includes the notion of process and multi-tasking, the memory management, the file system, the security and the power management functions.

Finally a conclusion of the relevant underlying operating system functions of a Virtual Machine or Interpreter required for the different classes of TPD is proposed. In completion, an operating system architecture proposition is done based on the resource management conclusions.

## 2. Existing Operating System Concepts

---

The Draft Software Architecture of the Trusted Personal Device (TPD) looks to the next 10 years in the future of smart cards. This document studies the state of the art of Operating System smart card technology and also other devices with powerful hardware that could be the future TPD. The most relevant devices are Pocket device or PDA more known as Embedded Operating System.

The Interpreted language and Virtual Machine technology is the state-of-the-art for runtime environments that power interpreted language, sophisticate garbage collection and bring a strong security model. The industry largely deploys such Interpreted Language framework based either on

Sun's Java language and Java Virtual Machine or on Microsoft's .Net Framework ECMA Common Language Infrastructure (CLI).

These Interpreted languages offer a complete level of interoperability from one hardware manufacturer machine to another, because the Interpreter or Virtual Machine is a software program built for each dedicated hardware.

But the major issue in the design of these Interpreted languages and execution environment is the security. The Interpreter or Virtual Machine check entrusted code with a Code Verifier and also check the executed program at runtime.

Sun's Java and Microsoft's .Net technologies are widely deployed in the Server area, Personal Computer but their presence is also significant in embedded devices such as Pocket device, PDA or Smart Phone. The smart card industry finds a great opportunity in JavaCard technologies to bring card vendor interoperability while enhancing whole security of the smart card products, there is also a .Net implementation in smart cards.

From the *InspireD* project point of view, the Operating System Software Architecture should be technology independent and not related to the particular Sun's Java or JavaCard technology. The concept of Virtual Machine or Interpreter is the same from an abstract point of view; both Virtual Machine and Interpreter are executed on top of an Operating System, which abstracts and manages the underlying hardware. This operating system can be accessible with standardized API allowing the loading and the execution of the native application or simply present for designed purpose and managing the low level hardware.

The TPD operating system state of the art is organized to cover the need of an operating system that support interpreted languages trough Virtual Machine or Interpreter. In other words, the study covers the aspect of operating system in term of the Resource Managements that are commonly required by Interpreters.

The state-of-the-art in smart card are Java Card and also Multos. The Pocket device with Linux embedded operating system is also a representative choice because the Linux operating system provided all functions of what is called modern operating system from state-of-the-art point of view and also because all information is freely available. But Linux itself doesn't provide Virtual Machine as is done by Microsoft with its .Net Framework. The Microsoft .Net Framework runtime is an ECMA standard name Common Language Infrastructure (CIL) that can be implemented on any platform. It is why an overview of the .Net Framework is also present in the state of the art.

Prior to enter in the state-of-the-art study, an analysis of the general Operating System functions is done. The operating system performs two basically unrelated functions, extending the machine and managing resources.

The other important function of the operating system is to abstract the machine as a simpler software machine providing interoperability at the instruction set between different hardware platforms such as Sun Java Virtual Machine or Microsoft Common Runtime Environment (or .Net).

Then, a study of the general Operating System requirements is provided. The operating system, in most of the cases, is a very complex piece of software with a huge amount line of code due to the strong interaction between numerous hardware elements: processors, memories, I/O devices and buses implying that it is designed for a long time and the different hardware platform coming up with their new features. The envisaged requirements are: Extensibility, Portability, Reliability, Security and Performance.

Then an overview of the existing Operating architecture is provided. The three architectures focussed are:

- **Monolithic System:** the components of monolithic operating system are organized haphazardly and any module can call another without restriction. The operating system is written as a collection of procedures, each which can call any of the other ones whenever it needs to.
- **Layered System:** the simplest organization in operating system is the layered structure. In the layered operating system, the components are organized into modules and layers them one on top of the other. Each module provides a set of functions that other module can call.
- **Microkernel System:** A monolithic or layered system will consume lots of system resources; traditional operating systems offer wide variety of services most of which are accessed once a while. Loading of monolithic kernel will consume abundant amount of memory resource and at the same time increase the complexity of a system. For instance, Linux's file system offers a wide variety of services that are rarely used by a compute intensive application. Such services can be moved out of kernel and make available in the form of user subsystem servers. The remaining portion of the kernel is known as microkernel.

Finally, the main part of the document deeply studies four relevant existing Operating Systems, focussing on features relevant for the TPD OS: Processes Management, Memory Management, Security Management, Input / Output Management, File System, Power Management, System Calls, System Start Up and Dynamic Component Loading. The considered operating systems are the following: **Java Card, Multos, Embedded Linux, Microsoft .Net Framework.**

### 3. Conclusions

---

The TPD general software architecture design is focused to support the execution of Applications written in Interpreted Languages such as Java, Common Runtime Environment (CLI) and executed by a Virtual Machine or Interpreter.

The conclusion of the TPD Draft Operating System Software Architecture is a synthesis of the business requirements analysis and also the requirements of the two other work packages, the WP2.2 Communication Architecture and the WP2.4 Application Framework in terms of operating system functions. The synthesis of requirements is done per TPD profiles, because each TPD profile is very different in term of business requirements, form factors and consequently in hardware features as well. First the different resources management: Process, Memory, Security, Input/Output, File System, Power and Component Management are examined and the main issues are pointed out to be fulfilled in the final software architecture. And, secondly, general operating system architecture is proposed in order to be commented. At this stage, the Draft TPD operating software architecture specification is a Request for Comment (RFC), and the final architecture specification will done in the D4 "General Software Architecture" deliverable document.

The study of the selected set of existing operating systems allows sketching the outlines of the TPD's operating system which as presented bellows:

#### *Process Management*

There are different possibilities to realize concurrency in operating systems and, as a result of this analysis, there is no unique solution to realize it. However, there is a unique demand for a multitasking enabled TPD operating system. This multitasking capability can be used on the one hand to support concurrency on application level, e.g. via white threads in the TPDs Virtual Machine. On the other hand, real time and security features can be considered in the design of the multitasking system. In the next steps this design has to be defined based on requirements derived of the TPD features.

### *Memory Management*

Virtual memory and page swapping: The two most significant differences between the current smart card implementations and the Linux Embedded memory management scope relate to the virtual addressing and paging: while current smart card implementations have no external memory for page swapping and virtual addressing is only rudimentarily used, the full Linux implementation supports both, even for embedded systems. Several optimizations have been proposed and implemented to adapt the Linux memory management to the limitations of small embedded systems.

Mass storage memory and DMA: The “D16 Draft Hardware architecture study for TDP” identifies two use cases for an external Flash storage: to store data (e.g. multimedia content) and to store applications.

In the next step of defining the O/S memory management layer, the detailed service requirements of the selected runtime environment have to be collected and the hardware support that can be achieved has to be identified. Based on these conditions the design of the TPD memory management then can start.

### *Security Management*

The security management of the different operating system developed in the respective chapters is not a specific entity; it is present in the implement in the different management: Process, Memory, I/O and File System.

Different requirements are under security management: the access control to the TPD resources and the secure execution of and byte code verification of the applications by the Virtual Machine or Interpreter.

### *Input/Output Management*

The Input/Output management of the different operating systems developed can be summarized as follow: JavaCard and Multos support only communication I/O management according ISO 7816-4 standard whereas Linux supports two types of I/O management the block special and the character special I/Os. Linux I/Os management types cover all the hardware devices (such as network communication, terminal, masse storage) and are exposed with the Linux File System Interface. The underlying security off the I/Os is implemented by the File System that expose the I/O device.

The above list of I/O management doesn't allow a conclusion with a clear status for the TPD I/O management, but opens up some questions that have to be analyzed.

### *File System*

As there is no clear requirement from the application framework (see TPD document D7) the TPD O/S will **not** provide an interface to the upper layers for managing files and directories. As there will be basic methods available to manage memory objects this still allows for the option of implementing file system functions in the application framework tailored to a specific use case if required later.

### *Power Management*

The operating system Process Management when the TDP scheduler has to manage the CPU clock during lengthy idle time detected.

The other type of chip set power management such as co-processor is under the responsibility of the application programmer or the API of the application framework to provide power management APIs.

### *Dynamic Component Loading*

The dynamic component loading capability is a common feature of the Interpreted Languages such as Sun Java runtime and Microsoft .Net or CLI runtime. This feature is a common part between the Application Framework and the Virtual Machine or Interpreter. The security of the components downloading and the trustfulness of these components have to be the managed by the Application Framework while the loading of the code itself is done by the Virtual Machine or Interpreter.

*Virtual Machine or Interpreter*

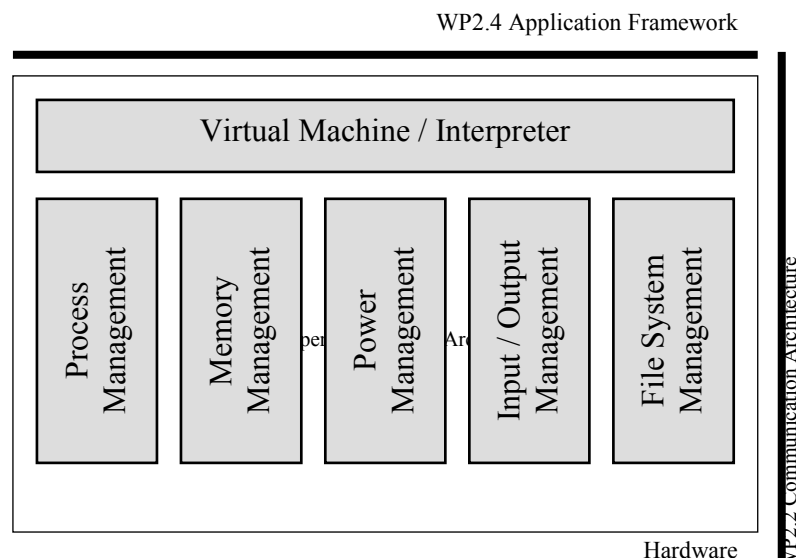
The Virtual Machine or Interpreter depends on the underlying technology; Sun Java Runtime Environments (JRE) specifies a Virtual Machine when the word of Interpreter is more a generic concept for all interpreted language like the Microsoft .Net or CLI runtime environment.

The requirements of Virtual Machine or Interpreter are:

- The secure execution of the interpreted language with type safe checking for example
- The byte code verification to detect corrupted instruction
- The Garbage collection of object that are no longer referenced

**TPD Functional Architecture**

The TPD Architecture is a functional architecture of the TPD operating system for the time been where the Virtual Machine or Interpreter is on the top of the different resources management functions: Process, Memory, Power, Input/Output and file system. The next task is the integration of the two other work package WP2.2 Communication Architecture and WP2.4 Application Framework architecture to provide a unified TPD software Architecture.



**Figure 1: TPD Operating System Functional Architecture**

## 4. Bibliography

---

- [BELS] Building Embedded Linux Systems, K. Yaghmour, O'REILLY ISBN 0-596-00222-X
- [GALU] Guide to Generating Application Load Units, v2.5.1 <http://www.multos.com/default.cfm/loaddoc.368>
- [IA64] IA64 Linux Kernel: Design and Implementation, D. Mosberger, S. Eranian, 2002, Prentice Hall, ISBN: 0130610143
- [JCRE03] Runtime Environment Specification. Java Card Platform, version 2.2.1. October 2003
- [JVM03] Virtual Machine Specification. Java Card Platform, version 2.2.1. October 2003
- [LXDD] Linux Device Drivers, A. Rubini, J. Corbet, 2<sup>nd</sup> Edition, 2001, ISBN 0-59600-008-1

- [M32R] Porting Linux to the M32R processor; Hirokazu Takata, Naoto Sugai, Hitoshi Yamamoto; Reprint from the „Proceedings of the Linux Symposium“ July 23th–26th, 2003, Ottawa, Ontario, Canada
- [MDG] Multos Developers Guide, v1.3.0,  
<http://www.multos.com/default.cfm/loaddoc.368>
- [MDRM] Multos Developers Reference Manual, v1.4.2  
<http://www.multos.com/default.cfm/loaddoc.368>
- [MMinLX] Memory Management in Linux, Desktop companion to the Linux Source Code, A. Nayani, M. Gorman, R. de Castro
- [MOS] Modern Operating System 2<sup>d</sup> edition, Andrew S. Tanenbaum; Pentice Hall ISBN 0–13–092641–8.
- [SimpleMP] Simple Memory Protection for Embedded Operating System Kernels; Frank W. Miller, Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, Monterey, California, USA, June 10–15, 2002
- [VarRadix] Variable Radix Page Table: A Page Table for Modern Architectures; C. Szmajda, G. Heiser, *Proc. 8th Asia–Pacific Computer Systems Architecture Conference*, Aizu–Wakamatsu City, Japan, September 23 – 26, 2003
- [VMMLX] Understanding the Linux Virtual Memory Manger, M. Gorman; Upper Saddle River, New Jersey 2004, Prentice Hall, ISBN 0–13–145348–3

## 5. Glossary

---

API	Application Programming Interface
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
ECMA	European Computer Manufacturer Association
I/O	Input/Output
ISO	International Standards Organization
JRE	Java Runtime Environment
MMC	MultiMedia Card
OS	Operating System
RFC	Request For Comment
TPD	Trusted Personal Device
VM	Virtual Machine