# New architectures for smart cards : the OCEAN approach

Olivier Caron, Vincent Cordonnier, Philippe Durif, Georges Grimonprez.

RD2P - Hôpital Calmette - Rue du Professeur J. Leclerc - 59037 Lille Cedex
Tél: (33) 20 44 60 46 - Fax: (33) 20 44 60 45 - Email: olivier@rd2p.lifl.fr

***Abstract** : There are today many and important reasons for giving future smart card generation new processors. This requirement is the result of more sophisticated applications that have to respect specific constraints such as response time and memory capacities.*
*In a smart card, hardware and software are closely interdependent because the code is stored in a rom at the production level. Thus, both hardware and software must be realized in an unique design process.*
*OCEAN project (Outils de Conception et d'Evalution d'Architectures Nouvelles) has been proposed to respond to that constraint. It bring the possibility to obtain a global evaluation of the chip and its embedded software. The project comprise the OCEAN Hardware description language, a specific C compiler to produce the application, a generic translator and an evaluation tool.*
*OCEAN is a part of the ESPRIT CASCADE project.*

## 1. The existing smart cards

The smart card domain is very specific and demanding. So, we present in this section the different hardware and software characteristics. These characteristics imply new ways of smart card application design.

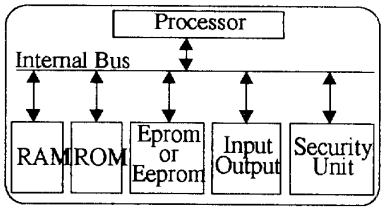
### 1.1 Hardware characteristics

The silicon component called the **micro-module** (see figure 1 on page 1) contains the following elements :

- The microprocessor.
  Used microprocessors are relatively old ones (Intel 8051, Motorola 6805, SGS Thomson ST 9). These processors have been choose for their size. There have not been specifically designed for the smart card applications.
- The ROM memory (Read Only Memory).
  When receiving the power supply, the microprocessor executes an unique program which is stored in its ROM memory. This program is called the MASK or the ROM CODE and looks like both the operating system and the application code.
- The RAM memory.
  This memory is both used for the variables of the mask and input/output buffer.
- The EPROM memory (Programmable Read Only Memory) or EEPROM memory (Electrically Erasable Programmable Read Only Memory).
  This memory is used for the permanent data management.
- Security unit.
  it's a hardware security unit which is checking the power supply and, in some cases, making a hardware protection of the different memories.

These components communicate with each other through an internal 8 bits bus. The external access is performed through a serial device equivalent to a serial asynchronous port. External services such as clock, power supply (etc...) are also available on the card connector.

**FIGURE 1.** Architecture of a micro-module




Memory capacities are very small. The following table show typical capacities for a "big smart card" :

**TABLE 1.** memory capacities

| ROM | RAM | EEPROM |
|---|---|---|
| 10 K bytes | 256 bytes | 8 Kbytes |

### 1.2 Software characteristics

The used masks must have the following functions :

1. Operating system functions.
   The mask must manage the input/output (protocol,...), EEPROM memory dynamic management.



0-8186-6315-4/94 $04.00 

2. Security.

   To ensure that the card and receiving device are communicating with each other, rather than with a fraudulent device inserted in the communication path, it is possible to use cryptographic algorithm.

3. data management

   The mask has in charge of the structure of the different data stored in EEPROM [19] [13].

### 1.3 Consequences

The above characteristics must be taken into account :

- The mask must be guaranteed as true to the specifications of the application because it is stored into a ROM. Removing a single default implies the destruction of all the smart cards.
- Software design decisions are determined by hardware characteristics. The first constraint is a static one : the mask must be compact (the ROM memory capacity is about 10 K bytes). The second constraint is a dynamic one : applications must optimise management of these variables that are stored in either registers or RAM memory (the RAM memory capacity is about only 256 bytes in the best case).
- At the moment, smart card applications are written in assembly language for memory size reasons and performance reasons.
- According to the memory capacities, the operating system is merged into the application software.

## 2. The future smart cards

Because they have to face increasing requirement and functionalities (Encrypted T.V., new generation banking cards, multiapplicative cards,...), future smart cards require new ways of conceptions. Moreover, future smart cards accentuate the physical inadequacies of the available tools [9] [10]. After a brief presentation of future smart card applications, let us present the consequences on hardware and software.

### 2.1 Software evolutions

The needs for new functions in the following domains are [12] :

1. Personal identification security

   The usual technic uses a number called the PIN (Personal Identification Number) assigned to a cardholder which is used by the cardholder to verify to a system that he can legitimately use a card. Identification, through the use of a PIN, can only prove that someone knows the key to the system. It does not prove that the person using the card is the authorised cardholder. For example, somebody who has difficulty remembering the PIN, and has written it on the card, provides all that is necessary for the card thief to gain access to the system. The use of biometric technics is a good alternative to this problem (fingerprint, dynamic signature, voice recognition, retinal pattern,...).

2. The operating system

   The need for model of multi-applicative smart cards implies new definition of operating system for smart cards [21].

3. Integration in information system

   Future smart cards as full partners of the information system, will be imply in the network configuration and will be able to lead transactions [16].

### 2.2 Hardware consequences

These software evolutions enable to define a possible hardware evolution [11] [12] [22] at two different levels :

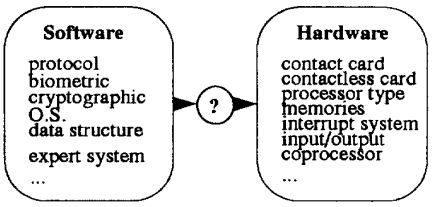
- Evolution of the microprocessor because some applications like cryptographic or biometric applications needs a lot of power of calculus.
- Evolution of the microprocessor environment. According to future applications (Operating systems, biometric, ...), it becomes necessary to put the following new elements in a smart card :

  - an interrupt system.
  - input/output ports.
  - Memory Management unit (MMU) or Memory Protection unit (MPU).
  - coprocessors (one example: a cryptographic unit).
  - a timer.

## 3. The OCEAN system

According the software and hardware evolution, there is a need for new ways of conception. The new design will be able to evaluate the appropriateness between the software and the hardware. The high constraints of a card make that step in the design process very important. Furthermore, the increase of hardware possibilities make that the choice of the best appropriateness (see figure 2 on page 2) will be an important step in the design process. This step will be able to simulate such innovations and then to evaluate the interest according the given software.

**FIGURE 2.** research of the appropriateness between hardware and software solutions




The actual system (the smart card applications are written in assembly language) become obsolete. It is now

imperative to have new methodologies of conception and adequate tools to design an evoluate smart card application according to a given hardware [5] [11].

### 3.1 The O.C.E.A.N. approach

We propose the **O.C.E.A.N.** approach (**O**utils de **C**onception et d'**E**valuation d'**A**rchitectures **N**ouvelles) which provide different tools. These tools allow both the conception and the evaluation of smart cards ( both mask and architecture). According to the different constraints of a card, the O.C.E.A.N. system permit to reach the three following goals :

1. The conception of a mask.

   The design tools of a mask must facilitate the writing of an operating system and the quality of the generated code (the mask is stored in ROM).
2. The evaluation of the micromodule.

   It is very important to simulate a card in the real conditions.
3. The evaluation of the appropriateness between the mask and the micromodule.

   This step is very specific at the world of smart card. So, there is an additional step in the process which serves to test the possibility of implementation on the real card chip. The available hardware must be closely connected with this program, by just using the minimum of resources (Registers, RAM, ROM).

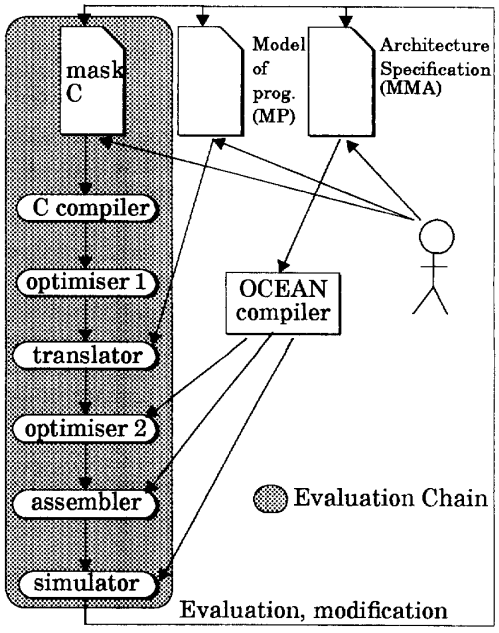
One of the great interest of the OCEAN system is that these three goals are not considered separately.

The three goals are in fact closely connected. For example, the results of the appropriateness between the mask and the micromodule (goal 3) may question either the mask (goal 1) or the micromodule (goal 2). So, to find the best result, we need to have an unique structure which join the three objectives together. The following figure (see figure 3 on page 3) show the principle of the OCEAN system.

In this figure, it appears an important point : the interactive side of this structure. The design process is in fact relatively simple. The first step consist in to define three statements : the mask, the model of programming and the architecture of the micromodule. The second step consist in the simulation of the mask in the real hardware conditions defined by the described micromodule. The third step consist in to provide the different measurements to evaluate the software/hardware set. These measurements can lead to two kinds of decisions :

1. The measurements show that we succeeded in all the goals : the execution, the response time, ..., size of the code are correct. In that case, the design process is finished.
2. The results show that the software/hardware set is either impossible (for example, the execution of the mask need more RAM memory than available) or the different goals are not correctly reached (bad response time, bad EEPROM memory management,...). In that case, we need update one or more of the three statements : the mask, the model of programming or the architecture.

**FIGURE 3.** Principle of OCEAN




According to the high constraints of a card, it appears that the research of the best appropriateness implies a lot of modifications of the three statements. That's why, the main characteristics of the OCEAN system are :

- *easy to modify* the software or the hardware statements.
- *easy to generate* new simulations and provide new evaluations from any modification.
- *quality of the simulation* which allow to the designer to evaluate precise critical sections of the mask.
- *precision of the evaluations* which allow to the designer to find easily the different problems.

All these measurements require a chain of tools going from the C description of the application to the delivery of the evaluation. The observation of the results can lead to three kinds of decisions : to update the mask (the program stored in ROM), the model of programming (MP) and the Micro-Module Architecture (MMA) statements.

By model of programming we mean the set on statements about the software eligible role of registers, the permanent allocation of the resources and more generally the programming environment [8].

Designing the MMA implies:

- The instruction set
- Registers which are used by these instructions
- Pipe-line structure that can be managed by the compiler.
- The different memories.
- the interrupt system.

When the MMA and MP statements have been changed the four last modules of the evaluation chain have to be redefined. The process will be repeated until results look satisfactory. Thanks to OCEAN structure (see figure 3), the designer can only concentrate on the mask, the MP and the MMA statements. From these statements, OCEAN compiler entirely generate a new evaluation chain described below.

The evaluation chain comprises the following modules: a C compiler which generates an intermediate code independent of the target machine, a high level optimiser working on that code, a translator from the intermediate code to the code of the target machine, this module needs to receive a precise definition of the MP and the instruction set of the target machine, a target machine oriented optimiser with two roles : to optimise the code according to the instruction set and the pipe-line management : instruction reordering, delayed branch, an assembler which has to provide static measurements and a simulator which has to provide dynamic measurements.

The C compiler is a specific one called the C_Card compiler. This compiler generates a code for a virtual machine called QUAD. [7] and [8] describe the C_Card compiler and the used methodology to translate the QUAD code to the code of the target machine and it will not be discussed in this paper.

## 3.2 Definition of the architecture of the MicroModule (MMA)

The OCEAN compiler produces the evaluations tools for the target machine. It receives as an input a description of the MMA for this target machine. The description language is an original one. It has been especially defined for the OCEAN project. A complete description of that language is available in [5]. The great interests of that structure are :

- A clear description of the architecture of the micromodule. The high level side of that language allow to describe easily the different elements of a micromodule.
- easy to generate a new evaluation chain. The designer can easily modify one characteristic of the micromodule. It is important for the interactive side of the OCEAN system.
- to define the different needs. The several simulations with different architectures of a micromodule allow to define the different needs (performances, size of memory resources,...).

### 3.2.1 The O.C.E.A.N. language and the CHDL languages

A lot of hardware description languages already exist : the CHDL languages (Computer Hardware Description Language). Let us cite the LIDO language [2] and the VHDL language [20] [18]. These languages are located at Register Transfer Level, they offers the possibility to define an architecture at a **functional** level and at a **structural** level more near the hardware realisation [3].

We consider the OCEAN language like a prototype language imposed by the constraints of a card. This language is closely connected with the software applications and the model of programming. This language do not replace the CHDL languages. The use of this language allow to define some important hardware characteristics.

### 3.2.2 The characteristics of the O.C.E.A.N. language

The following characteristics must be taken account into the OCEAN language : a simple description, the possibility to describe RISC processors, the description of the processor environment and the generation of simulation tools.

According to these characteristics, the OCEAN language must define the architecture of instruction set (registers, instruction format, instruction semantic), the structure of the pipeline, the memory resources and the interrupts.

The following example illustrates this language with a RISC machine : the MIPS architecture.

### 3.2.3 Register Description.

Most of the RISC architecture use two families of registers : basic registers and windows of registers. The OCEAN language supports these two possibilities. The PC register must be specified.

**TABLE 2.** register description

| MIPS registers | OCEAN Description |
|---|---|
| Architecture 32 bits<br>general registers<br>:R0..R31,<br>mul/div registers :H,B.<br>instruction pointer : IP. | **Machine**[a] mips:32;<br>**Registers**<br>R[32]:32,H:32, B:32,<br>IP:32 ;<br>**PC is** IP ; |

a. KeyWords are in bold

### 3.2.4 Pipe-line description.

According to the type of the instruction the pipe-line will decompose the execution in various sets of different

steps. Every step can comprise one or more elementary actions.

**TABLE 3.** pipe-line description

| MIPS pipe-line | OCEAN Description |
|---|---|
| **UAL instruction :** | **Pipeline** UAL : |
| stage 1 : Read Instruction | ( **Read_Inst** ) , |
| stage 2 : Decode, Read Registers, increment pc | ( **Dec_Inst Read_Reg Inc_pc** ) , |
| stage 3 : execute | ( **Exec_Inst** ), |
| stage 4 : nothing | ( ), |
| stage 5 : Write Result | ( **Write_Reg** ) ; |
| **Jump Instruction :** | **Pipeline** Branch : |
| stage 1 : Read Instruction | ( **Read_Inst**) , |
| stage 2 : Decode, Read Registers, compute branch adr | ( **Dec_Inst Read_Reg Inc_pc** ), |
| stage 3 : Jump | ( **Update_pc** ) , |
| stage 4 : nothing | ( ), |
| stage 5 : nothing | ( ) ; |
| **Load Instruction :...** | **Pipeline** Load :... |

### 3.2.5 Instruction format description

This section describes the various possible types of operands for each format and their internal code. Registers are specified as used for input or output.

Instruction Formats MIPS :

Operate 1 Format

| 6 | 5 | 5 | 5 | 5 | 6 |
|---|---|---|---|---|---|
| Opcode | S1 | S2 | D | v1 | code |

Operate_Imm Format

| | | | |
|---|---|---|---|
| Opcode | S1 | D | imm |

Branch1 Format

| | | | |
|---|---|---|---|
| Opcode | S1 | D | disp |

Branch2 Format

| | |
|---|---|
| Opcode | disp |

OCEAN description :

**Format** Operate1 : ( **Register out** D **Register in** S1,S2 **Value** v1,code)

(**opcode**:6 S1:5 S2:5 D:5 v1:5 code:5) ,

**Format** Operate_Imm : ( **Register out** D **Register in** S1 **Value** imm)

( **opcode**:6 S1:5 D:5 imm:16) ,

**Format** Branch1 : (**Register in** S1,S2 **Offset** disp)

( **opcode**:6 S1:5 S2:5 disp:16),

**Format** Branch2 : ( **Offset** disp)

(**opcode**:6 disp:26) ;

### 3.2.6 Instruction description

This section links for every instructions the descriptions about format, pipe-line management and contains a semantic description of the activity of the instruction. This description is written in C.

**TABLE 4.** Instruction Description

| MIPS Instructions | OCEAN Description |
|---|---|
| Instruction ADD, OPeratel Format, opcode : C_ADD, Pipeline type : UAL | **Instruction** ADD : (Operatel C_ADD UAL), **begin_c** **AluOut** = S1+S2 ; **end_c** ; |

### 3.2.7 Memory description

This section define the different used memories in the micromodule. More precisely, this section describes :

- The type of the different used memories (ROM, RAM,...)
- The size of each memory.
- The available address for each memory.
- The access rights for each memory ( read, write, ...).

The following table give an example of a memory description :

**TABLE 5.** Memory description

| OCEAN Description |
|---|
| **MEMORY ROM** : |
| ($0000 **to** $27FF) , |
| ( **READ_INST READ_MEM** ) ; |
| **MEMORY EEPROM** : |
| ( $6000 **to** $7fff : 1), |
| ( **READ_INST READ_MEM WRITE_MEM**); |
| **MEMORY RAM** : |
| ( $FEC0 **to** $FFBF), |
| (**READ_MEM WRITE_MEM**) ; |

The available elementary actions (see "Pipe-line description." on page 4) are specified for each memory. Thus, for this example, it is impossible to have an executable instruction in the RAM memory (the elementary action READ_INST is not specified for this memory).

### 3.2.8 Interrupt system description

This section allow to specify the different interrupts.This section enumerate the name, the type (direct or indirect) and the address for each interrupt. The designer must write, at different sections, the functioning of each interrupt.For this, OCEAN provides different standard variables and procedures to manage these interrupts.

Let us show with three types of interrupt the OCEAN description :

- An internal interrupt : *zero divide.*
- A hardware interrupt : a *timer.*
- A software interrupt called *user.*

**TABLE 6.** Interrupt system Description

| Interrupts | OCEAN Description |
|---|---|
| zero_divide interrupt effective address is $2020. | **INTERRUPT** zero_divide : ( **DIRECT** : $2020) ; |
| timer interrupt indirect address : $2008. | **INTERRUPT** timer : ( **INDIRECT** : $2008) ; |
| user interrupt indirect address :$200C. | **INTERRUPT** user : ( **INDIRECT** : $200C) ; |

The interrupt conditions are tested by the following ocean standard function :

int test_interrupt(int test, char name[MAX_NAME]) ;

This function start the interrupt called *name* if and only if the condition called *test* is true. This function return zero if the interrupt is not active.

In our example of internal interrupt zero_divide, the description of the DIV instruction manage this interrupt like this :

```
INSTRUCTION DIV :
( Inst_Reg 10 DivPipe ),
begin_c
  if (!test_interrupt(rs2 == 0,"zero_divide"))
    AluOut = rs1 / rs2 ;
end_c ;
```

Software interrupts are also described in the instruction section. Let us show with the interrupt called user :

```
INSTRUCTION INT :
( Inst_im 61 Ual),
begin_c
  if (im == 0x10) test_interrupt(TRUE,"user") ;
...
```

The management of hardware interrupt is realized by an additional elementary action called TEST_INTERRUPT. This action is executed at every machine cycle. This section must be written in C language in the last section of description of a micromodule : the section of *function and phase description.* Let us show with the timer interrupt :

```
PHASE TEST_INTERRUPT :
begin_c
  test_interrupt(timer_test[1](),"timer") ;
end_c ;
```

The management of interrupt mask is realised by these two following functions :

int get_interrupt (char name[MAX_NAME]) ;

- void set_interrupt (int state, char name[MAX_US]) ;

*get_interrupt* return the value of the interrupt mask of the *name* interrupt, *set_interrupt* set this mask at *state* value. Thanks to these two functions, it is possible to simulate the different priority of each instructions.

### 3.3 The simulation module

The OCEAN compiler produces two other modules: An assembler with extra capabilities for the static analysis.

A smart card simulator with the usual facilities such as step by step mode, break points, trace mode. The simulator also provides tools for the dynamic analysis.

The simulation process provides the following information :

- Memory size for the used ROM and RAM.
- Number of used registers.
- Elapsed time for executing the application code. As there is no real time definition of the target machine, the result is only available as a number of elementary machine cycles. When a pipe-line exists, a cycle must be understood as the time required for one stage. It is quite easy to convert this information to a real execution time giving the clock rate of the target machine.
- Number of pipe-line suspension cycles and for each of them a description of the cause (Conflict on resources, conflicts on control, conflicts on data: read after write, write after write,...).

## 4. Illustration

We illustrate the OCEAN system with a short example. We have described, in the OCEAN language, a micro-module based on a 32 bits RISC machine : the DLX machine

1. test_timer() is described in C language.

which is presented in [17]. This machine is well representative of the last generation of RISC architectures. The application code is an algorithm of photo compression [1] which require a lot of calculus power.

The memory capacities of the described micro-module are :

- ROM : 10 K bytes
- RAM : 256 bytes
- EEPROM : 8 K bytes

We realise a first study with the following model of programming :

**TABLE 7.** study 1 : model of programming

| | |
|---|---|
| R[0] | = 0 |
| R[1]..R[24] | Variables of the C_Card program |
| R[25] | Return value |
| R[26]..R[28] | Temporary variables |
| R[29] | Stack pointer |
| R[30] | Base pointer |
| R[31] | Return address |

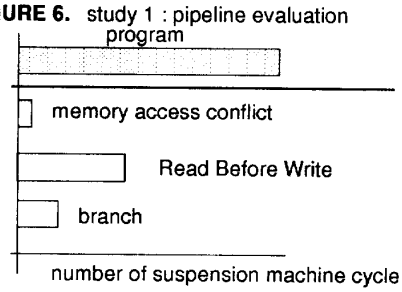
OCEAN evaluations revealed the following observations :

1. Because of technology, the data memory space is limited to 8 Kbytes then it does not require a 32 space address
2. The R[30] register is not used because the algorithm is not a recursive one
3. Only two of the three registers used for temporary variables are used
4. The Table 8 on page 7 show the need for the different memories.
5. The figure 6 on page 7 show an evaluation of the pipeline and the different suspensions of the pipeline ( memory access conflict, RBW (Read Before Write) and branch)
6. The MUL instruction take 30 percent of the program execution.

**TABLE 8.** study 1 : used memories

| ROM | RAM | EEPROM |
|---|---|---|
| 1,9 K bytes | 143 bytes | 4 Kbytes |

**FIGURE 6.** study 1 : pipeline evaluation




According to these observations, we decide to realise new simulations with the following modifications at different levels :

1. At the C_Card source level (study 2) :
   to replace MUL instruction by Left Shift Instruction when it's possible.
2. At the model of programming level (study 3) :
   The new model of programming is described in the Table 9 on page 7.
3. At the architecture of the micromodule level (study 4) :
   the new micromodule has got a data bus and an instruction bus.

**TABLE 9.** study 3 : model of programming

| | |
|---|---|
| R[0] | = 0 |
| R[1]..R[26] | Variables of the C_Card program |
| R[27] | Return value |
| R[28]..R[29] | Temporary variables |
| R[30] | Stack pointer |
| R[31] | Return address |

These new studies show several improvements :

- The study 2 is 8 percent faster than the first study (important decrease of number of suspension cycle (Read before write)).
- The study 3 is 7 percent faster than the first study and present a better compactness of code ( 1,6 Kbytes) and uses less RAM (120 bytes)
- The study 4 only is 1 percent faster than the first study.

We realize a last simulation which combines both the study 2 and 3. The different results are shown in the following table :

**TABLE 10.** results

| | speed (20 Mhz) | ROM size | RAM size |
|---|---|---|---|
| study 1 | 2,00 sec | 1,9 K | 143 |
| study 2 | 1,84 sec | 1,9 K | 143 |
| study 3 | 1,86 sec | 1,6 K | 120 |

**TABLE 10.** results

| | speed (20 Mhz) | ROM size | RAM size |
|---|---|---|---|
| study 4 | 1,98 sec | 1,9 K | 143 |
| study 5 | 1, 56 sec | 1,6 K | 120 |

This short example illustrates the interest of OCEAN as a general purpose tool for designing new applications and related hardware in the smart card area.

## 5. Conclusion - Perspectives

There is a direct use of the OCEAN system because OCEAN is a part of the European project called CASCADE (ESPRIT n°8670). The goal of this project is to define a new micromodule based on a 32 bits RISC architecture. The different partners are : Gemplus Card International : a smart card manufacturer, Texas Instruments, ARM (Advanced RISC Machine), Domain Dynamics Limited (for biometric applications), Neural Computer Sciences, UCL (Catholic University of Louvain - Belgium) for cryptographic applications and RD2P (University of Lille).

Our work consist in to find hardware or software solutions to introduce evoluate applications such biometric, cryptographic applications in a micromodule based on an ARM core risc. The use of OCEAN is an important element to define a new architecture of smart card for future evoluate applications. We have outlined the specific context of the card leading to OCEAN. There are many other domains where a microprocessor chip will always execute the same program (toys, printers, process control devices etc. ...). The OCEAN approach and methodology will probably benefit these domains.

## 6. References


[1] T. Alexandre, "La compression de données dans la carte à microprocesseur", Publication LIFL, décembre 1993.

[2] P. Bakowski, A. Pawlak, "LIDO-A Silicon Compiler Preprocessor", EUROMICRO Conf., Venice 1986.

[3] P. Bakowski, "Mode d'emploi de LIDO", Document LIFL, 1987.

[4] M. Barbacci, "Instruction Set Processor Specifications (ISPS) : The Notation and Its Applications, IEEE Transactions on computers, vol. C30, N°1, Janvier 1981.

[5] Olivier Caron, G. Grimonprez, "O.C.E.A.N. : langage de spécifications d'architectures ", Publication LIFL, juin 93.

[6] Olivier Caron, V. Cordonnier, G. Grimonprez, "OCEAN : a hardware and software tool for design of future smart cards", Barcelona, Euromicro'93.

[7] O. Caron, G. Grimonprez, "O.C.E.A.N. : A C compiler for new smart card applications", International Conference in Compiler Construction, CC'94, Edinburgh, avril 1994.

[8] O. Caron "Méthodologies de conception et d'évaluations d'architectures RISC adaptées aux futures cartes à microprocesseur", Thesis 1994.

[9] V. Cordonnier, G. Grimonprez, R. Beuscart, "Smart Cards and Portable Data Files: a glance at the future.", Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol 13, N03, 1991, page 1387.

[10] V. Cordonnier, "Smart Cards: present and future applications and techniques", Electronics & communication engineering journal, Octobre 1991.

[11] V.Cordonnier, "Assessing the future of smart cards", Proc. of the CardTech'92, Washington, DC, avril 92.

[12], J. McCrindle, "Smart Cards", IFS Publications/ Springer-Verlag, 1990.

[13] E. Gordons, G. Grimonprez, "A card as element of a distributed database", IFIP WG8.4 Workshop, Ottawa 1992.

[14] G. Grimonprez, P. Paradinas, "A new approach in code development: C-Card and Cossack", Card-Tech 1991.

[15] G. Grimonprez, "Etude et réalisation d'une carte à micro-processeur intégrée aux systèmes de gestion de bases de données", Mémoire d'habilitation, Février 1992.

[16] M.P. Haye, "The problematic of designing an information system using smart cards", Inforsid'92, Clermont-Ferrand, 1992

[17] J.L. Hennessy, D. Patterson, "Architecture des ordinateurs : une approche quantitative", Mc Graw-Hill.

[18] Proceedings of the IFIP WG 10.2 Ninth International Symposium on "Computer Hardware Description Languages and their Applications", Washington D.C., 19-21 Juin 89

[19] Normes internationales ISO, 7816-4, "Commandes d'échanges".

[20] S.P. Levitan, A.R. Martello, R.M. Owens, M.J. Irwin, "Using VHDL as a language for synthesis of CMOS VLSI circuits", Juin 1989.

[21] T. Peltier "Operating System for nomadic object",APPLICA'93.

[22] M. Ugon,"Le futur de la carte à puce", Cartes'93, Octobre 1993.