

faculty of computer science, iasi, romania  
<http://www.infoiasi.ro/>



## **old and new bind together** **connecting unix servers to siemens mainframes**

**project as bachelor of science**  
**candidate: radu filip**

**coordinators:**  
**louis colman (medior b.v.b.a., belgium)**  
**assistant professor sabin buraga**

iasi, june 28, 2002

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>The problem to be solved</b>	<b>3</b>
2.1	Background . . . . .	3
2.2	What is not an option . . . . .	5
2.3	Guidelines of a solution . . . . .	6
<b>3</b>	<b>The old: SNA &amp; CPI-C</b>	<b>7</b>
3.1	A little history: What is SNA? . . . . .	7
3.2	SNA Overview: Terms & Concepts . . . . .	10
3.2.1	The SNA protocols stack . . . . .	10
3.2.2	SNA Network types . . . . .	11
3.2.3	SNA PU Types . . . . .	12
3.2.4	SNA LU Types . . . . .	13
3.2.5	SNA Traffic . . . . .	15
3.3	CPI-C Overview: Terms & Concepts . . . . .	15
3.3.1	Conversation Types . . . . .	16
3.3.2	Send-Receive Modes . . . . .	16
3.3.3	Program Partners . . . . .	16
3.3.4	Node Services . . . . .	17
3.3.5	Side Information . . . . .	18
3.3.6	Program Flow - States and Transitions . . . . .	18
3.3.7	Program Calls . . . . .	21
3.4	Using TPS/SNA CPI-C implementation and Node Services . . . . .	21
3.4.1	Configuration files used . . . . .	22
3.4.2	Running TPS/SNA Node services . . . . .	22
<b>4</b>	<b>The new: GNU/Linux Operating System</b>	<b>25</b>
4.1	Why Linux? . . . . .	25
4.2	Linux software used in this project . . . . .	26
4.3	A quick note about SUN/Solaris . . . . .	27
<b>5</b>	<b>Our solution</b>	<b>28</b>
5.1	Why just CPI-C is not good enough? . . . . .	28
5.2	CPI-C Socket Overview . . . . .	29
5.2.1	Solution for existing applications . . . . .	30
5.2.2	Solution for future applications . . . . .	33
5.3	Using CISAM_DBCON module . . . . .	35

5.3.1	Unpacking the archive file . . . . .	35
5.3.2	Makefile configurations . . . . .	36
5.3.3	Compiling and linking library . . . . .	38
5.3.4	Installing library . . . . .	38
5.3.5	Uninstalling library . . . . .	39
5.3.6	Compiling and running test/sample programs . . . . .	40
5.3.7	Compiling, linking and other applications against this library . . . . .	40
5.3.8	CISAM_DBCON implementation overview . . . . .	41
5.3.9	Notes regarding server side partner program . . . . .	42
5.4	Using CPI-C Socket module . . . . .	43
5.4.1	Unpacking the archive file . . . . .	43
5.4.2	Makefile configurations . . . . .	44
5.4.3	Compiling and linking library . . . . .	44
5.4.4	Installing library . . . . .	45
5.4.5	Uninstalling library . . . . .	45
5.4.6	Compiling and running test/sample programs . . . . .	46
5.5	Programming with CPIC_Socket . . . . .	46
5.5.1	Class overview . . . . .	46
5.5.2	CPIC_Socket class public methods . . . . .	47
5.5.3	Creating epic_socket objects . . . . .	48
5.5.4	Creating a CPI-C conversation . . . . .	49
5.5.5	Sending data to mainframe . . . . .	50
5.5.6	Getting data from mainframe . . . . .	51
5.5.7	Ending a CPI-C conversation . . . . .	52
5.5.8	Set logging level . . . . .	52
5.5.9	Getting logging level . . . . .	52
5.5.10	Setting synchroniztion level . . . . .	52
5.5.11	Getting last error . . . . .	53
5.5.12	A sample CPIC_Socket program . . . . .	53
5.5.13	Note regarding C++ applications and TPS/SNA CPI-C implementation . . . . .	55
5.5.14	Notes regarding server side partner program . . . . .	55
5.5.15	Notes regarding further applications that might use CPIC_Socket . . . . .	56
5.6	Tasks for near future . . . . .	57
<b>6</b>	<b>Conclusions</b> . . . . .	<b>58</b>
6.1	General considerations . . . . .	58
6.2	Applicability of this project's solution . . . . .	58
6.3	Future developments of CPIC_Socket . . . . .	59
<b>A</b>	<b>Directory structure of attached CD-ROM</b> . . . . .	<b>61</b>
<b>B</b>	<b>About this paper</b> . . . . .	<b>62</b>

# Chapter 1

## Introduction

Since history of the human being begun, times ran faster every single year, months and day. The sunrise of the humanity took tens of thousands of years. Ancient times last only few thousand years. Middle Eve shrank to hundreds of years. It was followed by industrial age and it's amazing 20th century with so many achievements.

For information technology field, where there is a continuous race counted in mili-, micro-, nano- and pico-seconds, fifty years of history are equivalent with the entire history of mankind. In relative shorts period of times, technologies, companies and organizations appears, rose and not few of them disappeared forever in the night of one and zero. It seems from a point of view that the Moore's Law does not apply strictly for CPU's. The same crazy speed is seen in everything IT is touching: storage sizes, network speeds, memories capacity, resolutions and size of the screens, printing, webpages, number of e-mails exchanged, and so on.

Creativity of the human being never had so many opportunities to innovate like nowadays, when we can use the Internet with it's amazing tools: tools for communication like e-mail, instant messaging and on-line chat that created communities of peoples in all aspects of the living, tool for interactive presentation which is the web, who replaced almost completely the old fashioned books of Gutenberg, tools to spread and multiply data, tools to analyze and synthetize huge quantity of information, to name just few of them.

All these technologies succeeded to get peoples together, and gave creativity and ideas the best conditions to have a chance in the reality, either virtually or not. But in the same time, a single mistake can make an idea to disappear as fast as it born.

But all these amazing things that come form IT have a downturn. We need the technology available today, so we can take advantage of it. Companies, governments and organizations worldwide invests in the latest available technologies that soon become more or less obsolete by other new technologies. The problem is that the rules of economy do not care about the rules of the IT. Laws of Adam Smith do not care the laws of Gordon Moore. Long term profits are based on predictability and time, while IT today looks maybe like Universe in its first days: a place of countless experiments and ideas, a space of fast successes and failures. This gap between developments in economy and fast developing in IT makes the story more interesting and complicated.

In order to be profitable, a technology must be used periods of times longer that we are used to count in IT. In this period of time, the technology used may become more mature and stable. And for many companies this means two things: they are harder to replace, since they become a core part of IT infrastructure and even if a new technology in the same

area was born meantime, the profit from implementing the new one can be far less than the costs needed to replace the current one.

This situation acts like a break for IT development, which is forced to take care of backward compatibility, but maybe this break is needed in order to prevent IT to become a complete mess like the Babel tower.

This project is about such an "old" technology that needs to interact with new technologies. It's not only about backward compatibility; it's about bringing together old and new to create valid solutions for real needs, present of future.

The purpose of this project is to address such a problem, to identify and implement a viable solution.

Work on this project was done by Radu Filip (<http://socrate.tuiasi.ro/>), as his project as candidate to bachelor of science, at the end of four-year studies on "Alexandru Iona Cuza" University of Iasi, Computer Science Faculty, Romania (<http://www.infoiasi.ro/>). The development of this project was supervised and coordinated by:

- *Louis Colman*, Operational Manager, Medior B.V.B.A., Belgium
- *Jan Van Camp*, Software Development Manager, Medior B.V.B.A., Belgium
- *Sabin Corneliu Buraga*, Assistant Professor, Faculty of Computer Science, Iasi, Romania

## Chapter 2

# The problem to be solved

### 2.1 Background

Medior B.V.B.A., (Antwerpen, Belgium, <http://www.medior.be/>) is a 40 years old company specialized in developing integrated solutions for cargo transport companies, especially in European Union, but not only.

Companies specialized in cargo transport of consumer goods among countries from European Union use specialized software developed over years to automate, monitor and process shipments of crates of goods across transport and supply lines. Such software was introduced quite a long time ago, starting with '70s, when first commercial computers started to become affordable for medium and big sized companies.

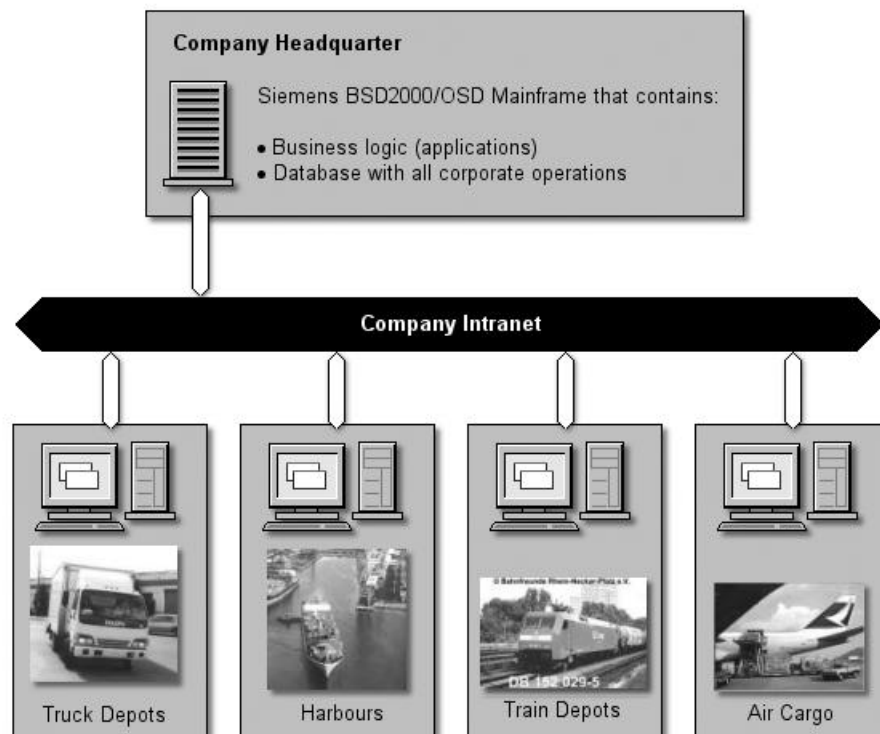


Figure 2.1: IT infrastructure of a cargo transport company

A number of such companies choosed to use and develop solutions based on Siemens mainframes. The classic architecture of such a model is presented in the figure 2.1.

Hardware and software details:

- Siemens Mainframes with BS2000 (now called OSD)
- Cobol applications written for OpenUTM transaction monitor on mainframe side
- Informix C-ISAM database systems
- SUN Unix servers with Solaris OS

Each cargo terminal consists mainly in a Control Center, powered with Unix server that coordinates activity within that area. Workers use wireless handheld scanners as mobile wireless terminals to know how to process incoming, outgoing or transit containers and how to fulfill and process orders for vehicles that arrives or depart.

The activity in such a cargo terminal is presented in the figure 2.2.

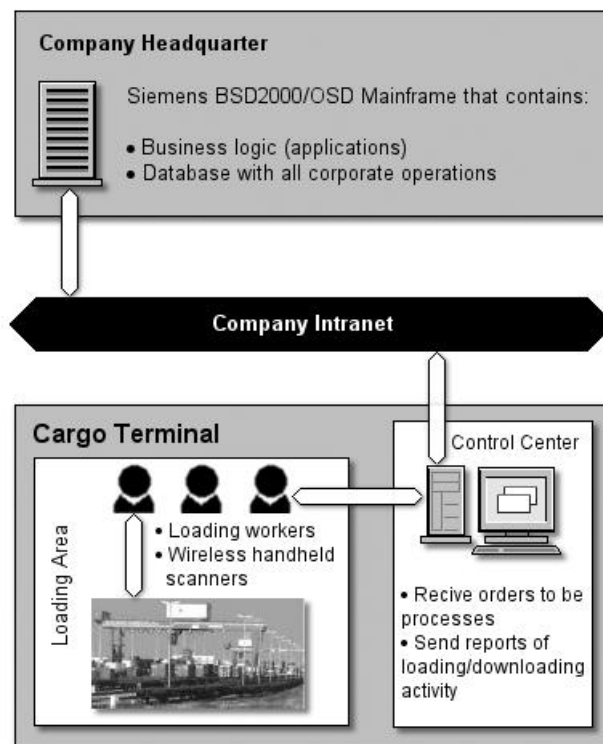


Figure 2.2: IT infrastructure inside a cargo terminal

Because of IT development stage at that time, and because the limitations of these products, some compromises was made. One of them was the lack of a dynamic data exchange protocol supported by the Siemens mainframes. Because of this, a number of workarounds had to be implemented. Specifically, applications developed on Solaris had to save data they want to transmit to mainframe in some local cache that periodically was uploaded on mainframe by some scheduled programs, over a file transfer link (FT-Link). Also, once a day, data from mainframe to Unix servers was transferred also via a FT-Link.

This approach solved somehow the initial problem, but introduced a number of other limitations, with secondary consequences in long term. The most important was a statically behavior of applications, because of the impossibility to exchange data in real time with a mainframe.

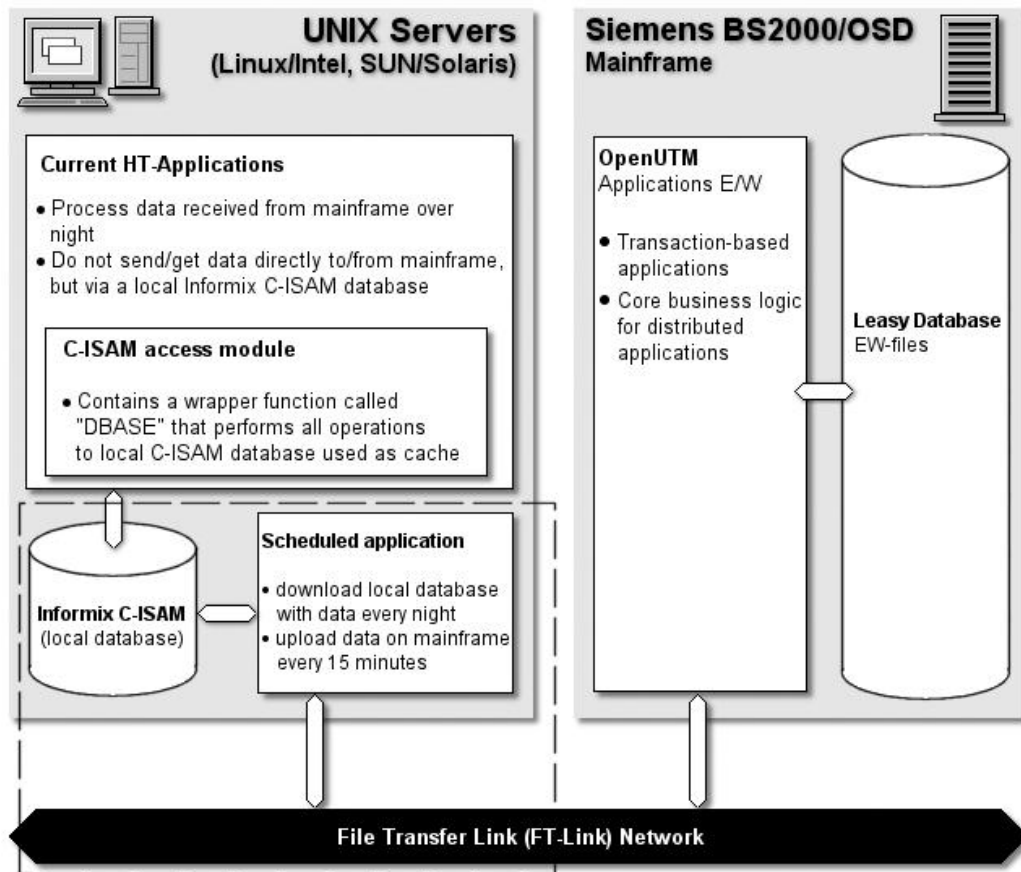


Figure 2.3: Application architecture

Figure 2.3 present the internal architecture of the applications that run on Unix servers and Siemens mainframe. The area bordered with dash-line emphasizes the place of the problem: non-dynamically data exchange with mainframe.

## 2.2 What is not an option

On the other side, there is a large installed base of mainframes and running applications, most of them which are mission-critical. These applications are very complex and was developed and polished in years. This means that mainframes and applications they are running are virtually impossible to be replaced without huge, unnecessary costs for organizations that use these technologies. It is far less expensive to train new generation of programmers to use and develop further current applications than going into adventure of replacing all installed base that works very well after so many years of continuous improvements and enhancements.

## 2.3 Guidelines of a solution

In year 2001 Siemens suggested that a possible approach to solve the problem of dynamic communication issue is to use on mainframe side a software component called *UPIC Carrier System*, supported in latest versions of their *OpenUTM* (Universal Transaction Monitor).

Starting from these facts, this project identify, explore and implement a solution that solves the problem of dynamic data exchange between Unix servers (Linux and Solaris) and Siemens mainframes. It also explores some possible future developments of this solution.

The following chapters of this project present what solution was implemented, in what way and how it can be used, by examples.

## Chapter 4

# The new: GNU/Linux Operating System

### 4.1 Why Linux?

It is not the purpose of this paper to address all things that can be said about this operating system that changes the way we develop software today. Started as a hobby in 1991 by Finish student (at that time) *Linus Torvalds*, Linux makes quick inroads into business markets. In the last three years it is considered by major data research companies like IDG (<http://www.idg.com/>) or Gartner Group (<http://www.gartner.com/>) the fastest growing operating system worldwide.

Although others free operating systems like BSD or Mach already existed already at the moment when it born, Linux become so successful because its developers took advantages of a number of technologies and facts:

- Commercial Unix market was fragmented between main players (IBM, SUN, HP, Digital, SGI, SCO); each of them added proprietary features that leads to incompatibilities between different implementations of Unix.
- Unlike for BSD kernel, Linus Torvalds choosed GNU Public License (GPL) for Linux kernel, which not only stipulate that the software is fully freely available, but any subsequent changes remains free software. This is a major difference compared to BSD style licensing, that allows companies to use BSD licensed software without returning their contribution back developers. GPL strength is that it keeps all the future developments free, as the original project.
- A large number of Unix-like applications like text editors, compilers, debuggers, shells, online manuals, system libraries were already developed under the GNU Project in 1991, and thus the Linux has a large application base since its very first days, a very important advantage. This is why we refer it as GNU/Linux. Also, X Consortium ported very quickly a free implementation of its X Windowing Operating System to Linux.
- The born of World Wide Web in 1991 and dramatic increase of Internet usage for masses worldwide, that allowed a big number of skilled youth to enter in contact with engineers worldwide and collaborate all together online in huge virtual communities not possible to exists before.

Let's analyze now the most important reason for why Linux adoption rate in business environments is so high.

- Linux made what X/Open or POSIX did not succeed: it made possible, based on its free nature, to have a single unified Unix specification and implementation, that runs on virtually all hardware platforms that exist today in the world.
- Also, it supports all standards that exist for, giving open standards an important, dominant role in industry.
- There is a huge installed base of commercial applications that runs on old Unix systems that need to be replaced in the near future, and Linux is the better choice because it requires minimum modifications to applications that need to be ported.
- It is not tied to a single vendor and it is actively developed without interferences from the evolution of IT markets and economy trends, in general.
- It is very robust, fast and secure, because of its free software development model and its *release early, release often* approach. It is not bug-free, no software is, but it has the fastest way to improve itself due to its open nature.
- It is cost-effective: instead of paying money for large volumes of licensed software (that often require the latest and most expensive hardware), Linux allows you to make important savings both in software (no licenses) and hardware. In the same time, customers can choose to buy, if they need and want, a number of related services like technical support and service, manuals, personnel training and certification, etc. This is the reason why free software is now chosen to be implemented by state administrations in a number of countries like Germany, France, China, Taiwan, Finland or Peru.

To use Linux in this project it was possible because existing SNA implementations for it. There are only benefits from this fact, because this way a modern and rich in features operating system can be integrated into a mature but not so widespread architecture, bringing new into a place condemned and stagnation otherwise.

We think that once again, that using Linux into this project proves in what way open and free technologies can be tailored and adapted to deal with existing mature technologies, bringing benefits to its users and opening new perspectives instead of mutual exclusion.

## 4.2 Linux software used in this project

The solution addressed by this project was developed on an Intel system with the following key software components:

- Distribution: Mandrake Linux 8.2
- Kernel version: 2.4.18-6mdk
- Gcc version: 3.0.4-3mdk
- GLibC version: 2.2.4-25mdk

- GNU Make version 3.79.1
- GNU ld version 2.12.90.0.3 20020323
- GNU bash, version 2.05.1(1)-release (i586-mandrake-linux-gnu)

Tests was performed on a similar machine with the very same software installed.

### 4.3 A quick note about SUN/Solaris

SUN/Solaris it is not Linux, and therefore should not be mentioned in this chapter. But from the "old" point of view, more exactly from three decades old Siemens technologies, Solaris and Linux can be both see as "the new". This is why this note was inserted into this chapter.

Founded in the beginning of 80's, SUN Microsystems succeeded to create one of the most used commercial version of Unix, *Solaris*. Because of that, Solaris is now one of the most spreaded non-free Unix in the world. Even if Linux was ported to Sparc processors in mid 90's, some of Solaris kernel key features like multi-threading and scalability on machines with a large number of processors (more than 8) preserver Solaris place among commercial Unix vendors. While SCO Unix, Digital Alpha, HP-UX or SGI Irix are almost history today, Solaris is still here and it will still be here for at least a number of years.

This is why the solution developed by this project need, finally, to available not only to a Linux system, but to a SUN/Solaris system as well. But for development stage of this project, only a Linux system was used.

Finally, we want to mention that even if the specification of any project leads to Solaris as the best choice, free software can still be there, because a large number of free software projects like gcc, bash (and almost all GNU software), Apache, KDE, GNOME and others was ported on Solaris.

# Chapter 5

## Our solution

### 5.1 Why just CPI-C is not good enough?

CPI-C means for SNA world what TCP means for Internet. CPI-C is rich in functions that can be used to build any kind of application that needs reliable ways to transport data from and endpoint to another. Then why just using CPI-C is not good enough? Despite it's great features, CPI-C has a number of down points, that are listed bellow.

1. There is no "unique" or "standard" CPI-C implementation. IBM developed SNA and CPI-C as an integrated, custom and proprietary solution, build around their family of mainframes (mainly S/390). IBM never intended to submit these protocols to independent standardizations bodies like International Standard Organization (ISO) or American National Standard Institute (ANSI). But IBM ideas about how to solve high-reliable communication between hosts located far away one from each other, inspired other corporations, which developed similar solutions and their own version of SNA (and CPI-C).

Siemens was one of these corporations who proceeded this way. But because no one of these implementations is free, open software or was not submitted to independent standard bodies, there are differences (or enhancements) between the implementation details of each one's CPI-C.

In case of this project, these incompatibilities do not stop here. Since no implementation of CPI-C for Linux was available from either IBM or Siemens (which provides only mainframe side of SNA implementation), we had to use a third-party product called TPS/SNA that contains a CPI-C version for Linux systems. They relay, of course, on IBM reference, but they don't have implemented all CPI-C calls and parameters.

This is why using just CPI-C in this project to solve the initial problem was quite difficult (although not impossible): every CPI-C call used had to be checked against three different set of documentations: IBM CPI-C Reference Manual, Siemens UPIC manual and TPS/SNA manual.

2. The second important aspect is that CPI-C protocol contains over that 60 possible calls. This is not too much, if we think to TCP/IP for example (which has about 50 calls), but in our case, we don't need all of them. These 60 calls was created to make CPI-C a protocol used in many various situations, while we need it for a certain group of specific applications which have a common pattern in the way they exchange data.

3. The third aspect results from the evaluation of the CPI-C implementation we used: TPS/SNA worked only in an Ethernet environment. This leads to the fact that Unix clients and Siemens mainframes must be in the same LAN, which has as a consequence very severe limitations. In order to avoid these, another layer above CPI-C must be considered, layer that should be accessible from other networks over TCP/IP protocol.

## 5.2 CPI-C Socket Overview

CPI-C Socket is our solution that tries to address all these problems. We choose this name as an analogy with BSD-like TCP sockets. We consider that this problem addresses all the problems regarding differences in CPI-C implementations and it's complexity.

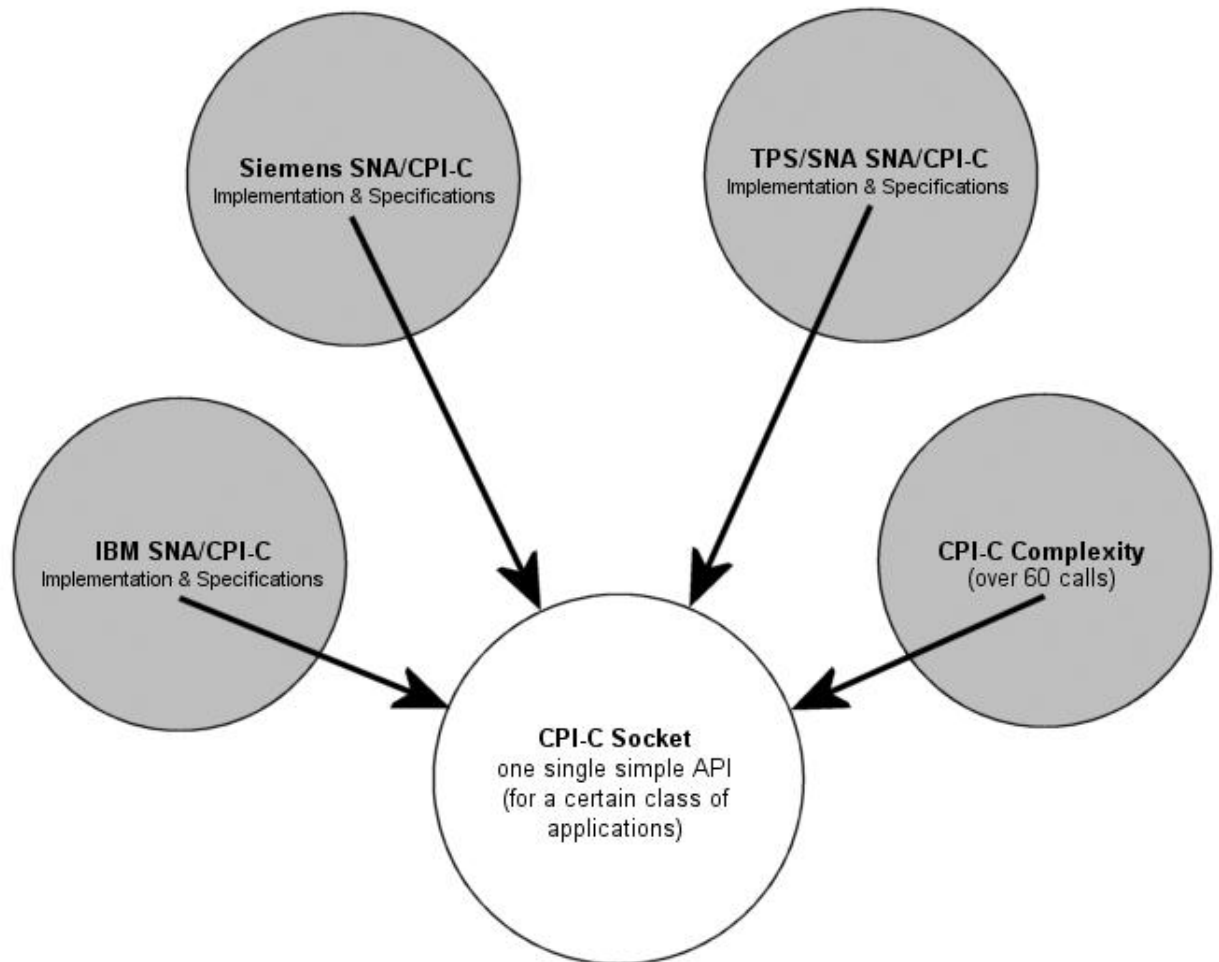


Figure 5.1: CPI-C Socket solution as the convergence of different implementations

### 5.2.1 Solution for existing applications

Let's look once again to the problem we want to solve:

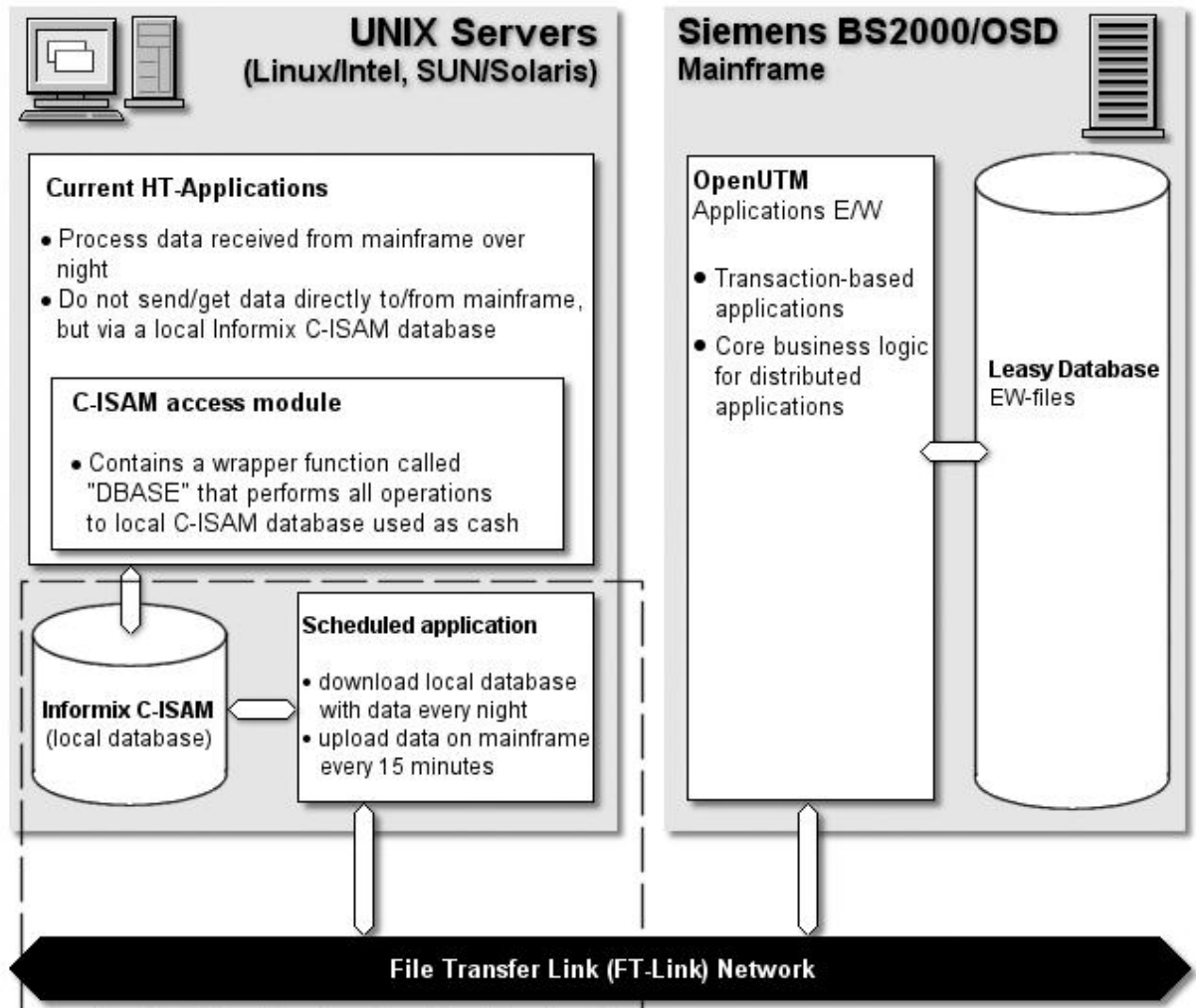


Figure 5.2: Current applications architecture, with problematic area surrounded by a dash-line border

We developed a module called `CISAM.DBCON`, which provides backward compatibility for current applications that uses only local data. These applications used a wrapper function called `DBASE` (there is no link with `dBase` RDBMS developed by AT&T/Borland) to store and retrieve data from local database. In `CISAM.DBCON` module we rewrote this wrapper function, so now every call that read or write data into local database is captured and send through network, via `CPI-C` to mainframe, like in the figure 5.3:

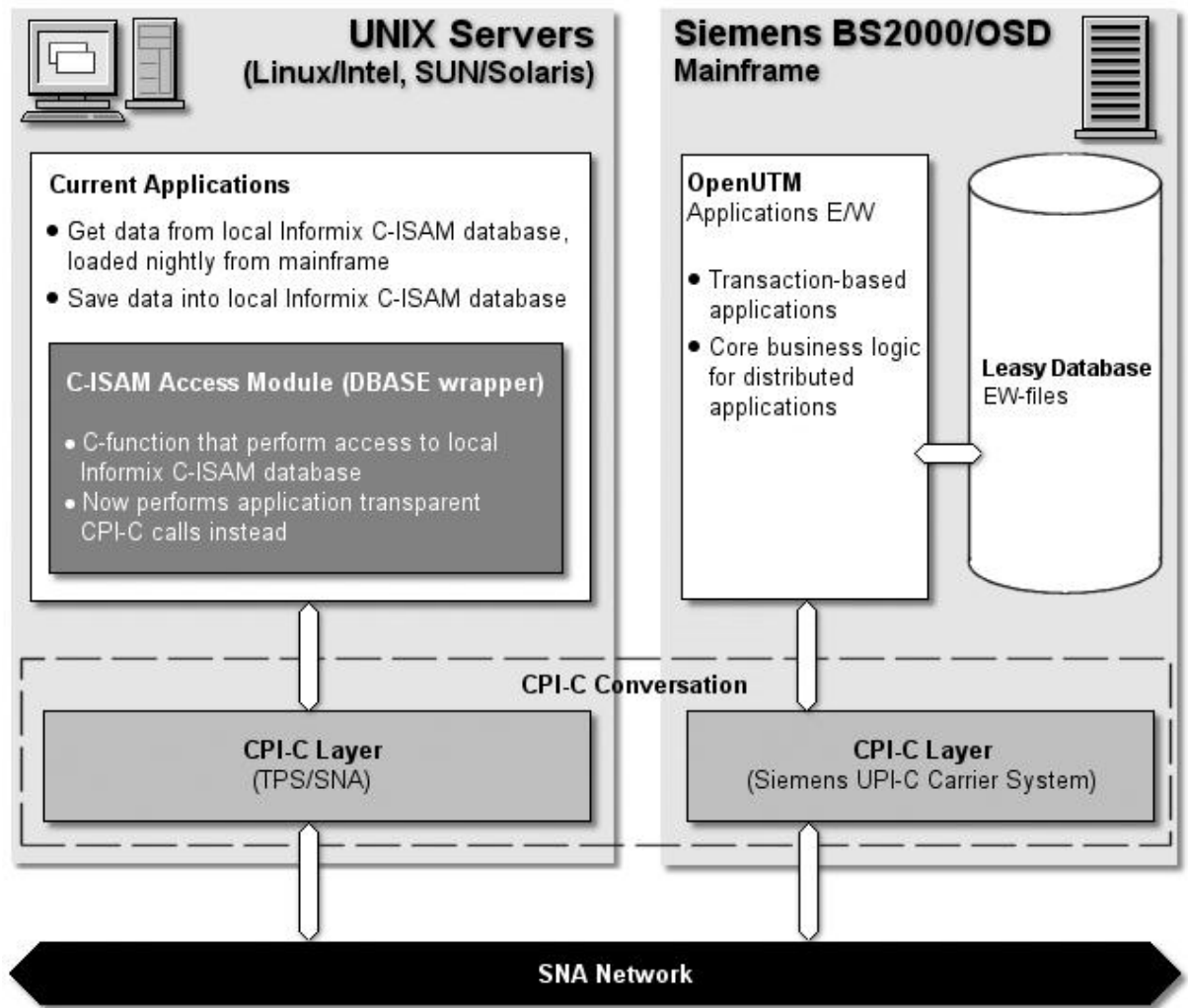


Figure 5.3: Current applications with rewritten C-ISAM module that uses CPI-C

This way, the current applications need not even to be recompiled, if they were linked dynamically, or need only to be recompiled, if they were linked statically. Figure 5.4 presents what happens internally and how the compatibility is kept.

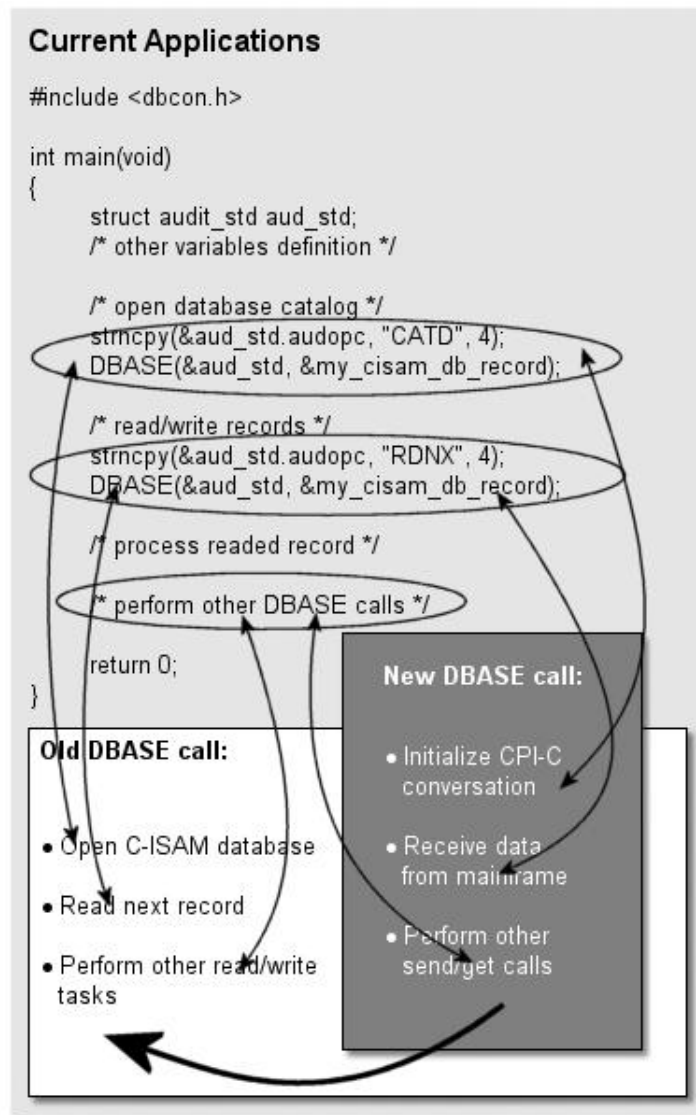


Figure 5.4: How new CISAM.DBCON module replaces old one while preserving current applications unchanged

In any case, the main benefit of this approach is that the current applications do not need any changes.

### 5.2.2 Solution for future applications

New applications that will be develop will don't have to use the rewritten wrapper DBASE function from CISAM.DBCON module. CPI-C Socket was designed to be used as a standalone CPI-C connector. From network layers point of view, CPI-C Socket is a new layer above CPI-C, as is seen in the figure 5.5:

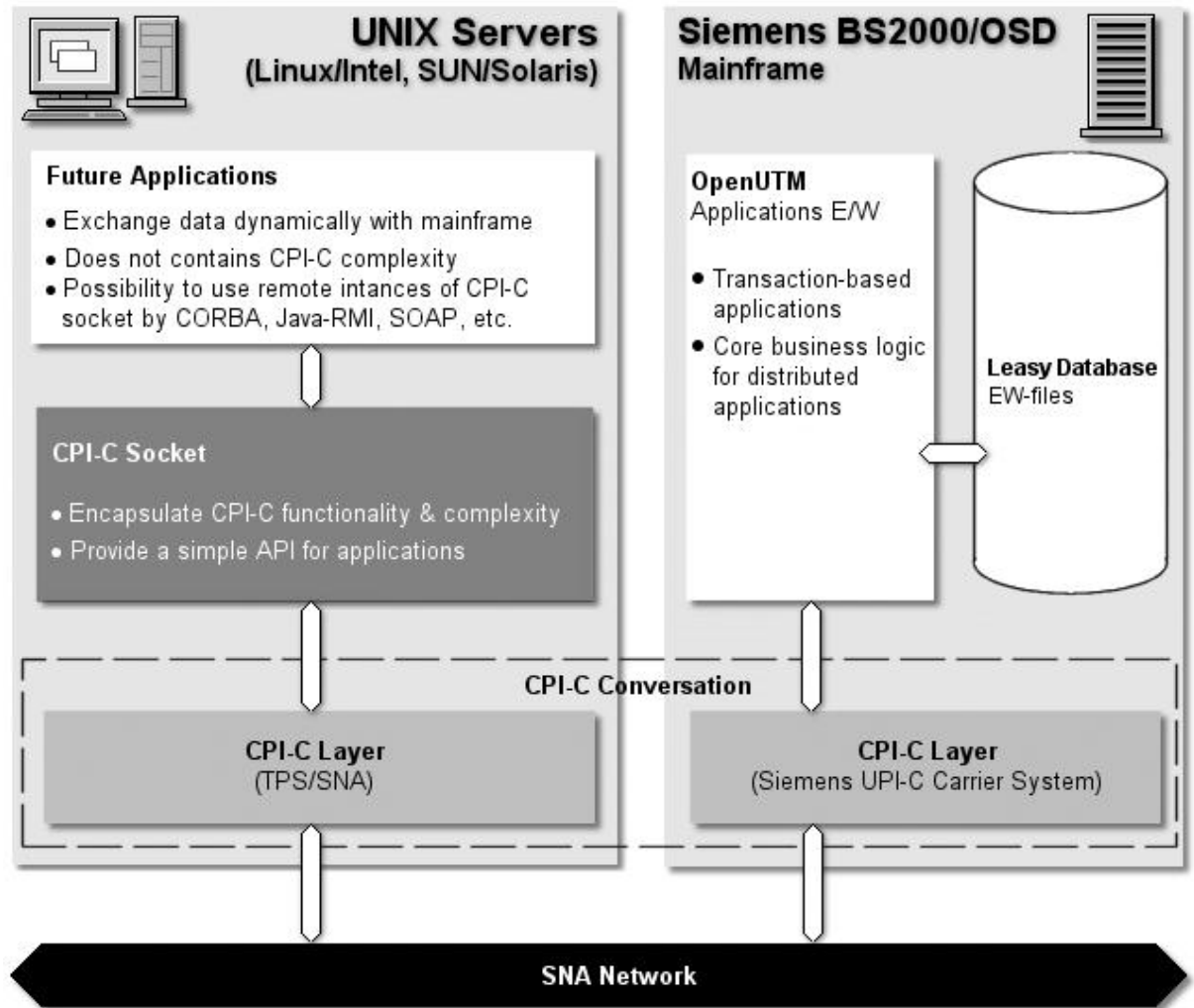


Figure 5.5: Network architecture of new applications, based on CPI-C Socket

From programmers point of view, CPI-C Socket it is a object oriented class that encapsulate CPI-C functionality and present developers a set of public methods (an API) that simplify interface to CPI-C. The figure 5.6 shows a sample CPI-C Socket skeleton application and represent the way CPI-C socket encapsulate CPI-C functionality.

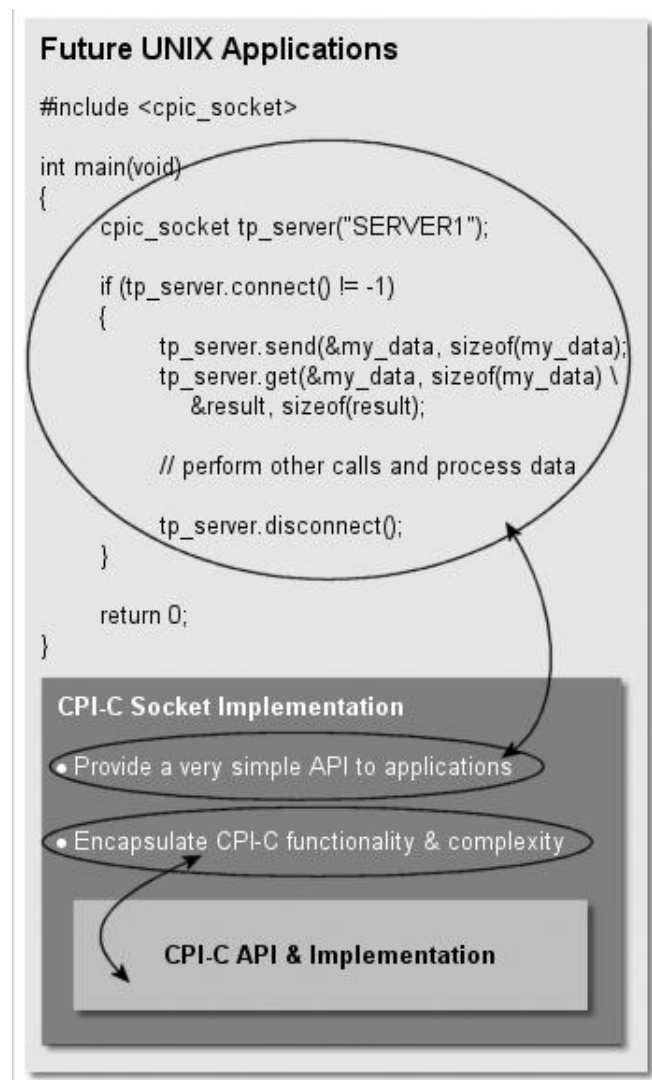


Figure 5.6: Sample new application that use CPI-C Socket

# Chapter 6

## Conclusions

### 6.1 General considerations

”*Old*” in IT does not necessary means *obsolete*. It can mean *maturity, stability* or *polished*. ”*New*” in IT does not necessary means *good* or *stable*. But neither of *old* or *new* cannot be ignored. For any real problem, both must be considered and ways must be found to get them to work together. Before working on this project I thought that TCP/IP networks are the only networking solution and all others were obsolete. Now I learned that TPC/IP is good for a certain class of problems. Other problems created other solutions. This is how SNA born.

*Old* and *new* can be connected together only if open protocols exists or at least some specifications exists. Without that, cooperation is not possible at all. Close standards are the real threat for any technology, no matter its age. For a solution provider, close standards leads to stagnation, disinterest and isolation. For a customer, close standard means dependency of vendor: when vendor die or abandon tat closed technology, customer has to start from zero.

### 6.2 Applicability of this project’s solution

On Medior B.V.B.A., Belgium, the solution proposed by this project and detailed in the previous chapters will be part of some projects like:

- Project Hand Held Terminals (currently running at this moment in Belgium and France, later also in Great Britain and possibly China)
- Delivery display: a global overview of incoming and outgoing trucks, dispatch system for loading trucks, assignment of the teams on the port area, etc.
- Data Warehouse: part of the upload tools for data from the mainframe to the DW; other directions
- Applications that require a graphical interface on the users side, like tools for management.

### 6.3 Future developments of CPIC\_Socket

Depending on the feedback that will result from deployment of this solution into production, and based on some predicted updated requirements from those who will use this solution, a number of possible development can be imagined:

1. **Specialized CPIC\_Socket classes:** Using C++ inheritance mechanism, CPIC\_Socket class can be easily derived, when needs for such a specialized version will occur. This will require from developers to write only the modifications for the new derived class, instead to fork code and maintain multiple libraries, as in case of standard C programming language.
2. **Interoperability:** CPIC\_Socket class can be easily transformed later into a component, available to any other application using CORBA. Any client application no matter the language in what language it's written (C, C++, Java, Python, Perl, etc) will can use our component using CORBA.
3. **Distributed applications:** CORBA also allows remote components: a component used by a CORBA application is not necessary need to run on the same machine as the application: application invoke the component and CORBA system transport the call to the machine where server1 component reside; that call is then processed, then CORBA transport back to the application the result of the call.

This could be very helpful to resolve the problem of CPI-C and routable protocols. We can have this component on a machine near mainframe that is communication with mainframe via CPI-C via Ethernet. Client applications that reside on different servers use the `cpic_socket` component from remote locations using CORBA, which is implemented over TCP/IP.

4. **Mobile clients:** with XML, WAP and others web-based technologies that make serious inroads today, we can easily have CPI-C proxies. This can consist in a WAP/Web server playing the role of middleware. Simple clients like mobile phones, or new generation wireless devices that knows WAP/Web can perform such calls to middleware platform that convert their call into a CPI-C call and when gets the response it forward it back to the mobile device.

# Appendix A

## Directory structure of attached CD-ROM

The attached CD-ROM contains:

- `modules/` directory contains the two modules of this project:
  - `cisam_dbcon-1.0.tar.gz`: backward compatibility CISAM\_DBCON module
  - `cpic_socket-1.0.tar.gz`: CPIC\_Socket module
- `papers/` directory contains:
  - `old_and_new.pdf`: complete project
  - `old_and_new.ppt`: presentation of the project
- `tps-sna/` directory contains:
  - `tpssna-20011211.tar`: TPS/SNA software used for CPI-C implementation and node services
  - `conf/` directory contains configuration files used for both server and client side of TPS/SNA
  - `servers/` directory contains various test and sample programs used as CPI-C servers

# Appendix B

## About this paper

This project was created using L<sup>A</sup>T<sub>E</sub>X. Figures was created using GNU GIMP graphic images processor, version 1.2. Conversions to EPS format was performed using Adobe Photoshop 6.0.

A webpage about this project exists on address:

<http://socrate.tuiasi.ro/studies/bachelor.php>