OSPORT - A Practical Tool for Porting or Adjusting Operating System Source-Code

Osvaldo de Souza, Helano S. Castro

LESC – Laboratório de Engenharia de Sistemas de Computação
DETI - Depto Engenharia de Teleinformática – Universidade Federal do Ceará (UFC)
Campus do PICI bloco 910 – Fortaleza – CE – Brazil
{osvaldo,helano}@lesc.ufc.br

Abstract. *Many projects frequently uses the "trial and error" approach for Operating System (OS) porting or maintenance, resulting in incomplete or inconsistent modifications. This can be partially explained due the absence of useful tools helping the source-code analysis, in order to determine what source-code must be modified for adjusting it to another hardware platform. In this paper, we present an application called OSPORT, based on a seminal method for detecting OS parts that should be adjusted in order to port or adjust the OS for a new hardware platform. The OSPORT provides: a complete list of source-codes that must be adjusted; the interdependence between these source-codes; the priority order of modifications for each source-code; and an effort-based schedule, in order to plan the modifications.*

1. Introduction

The Operating System (OS) is the most important software-element present in systems based on microprocessors or microcontrollers. However, it is regrettable that there are not many tools that support the designer when it comes to maintenance, much less when one needs to port that OS into a different hardware platform. The available methods and tools focus primarily on hardware elements and this can be partially explained by the fact that usually the design of Embedded Systems (ES) and Personal Computers (PC) are developed by engineers who are hardware-oriented designers [LEE 2000].

The software elements related to high-level applications consume most of the development efforts, while (OS) has minimal attention and frequently it does not figure as a priority item. Actually, OS adjustment efforts rarely are not improvised. Recent researches have proposed development models where hardware and software development is carried out in parallel. However, they do not provide any OS development or adjustment-specific approach [POSADAS 2004][CARRO 2004].

Although OS is a significant subject in computer science, as far as its architecture and design are concerned, there is no specific approach dealing with OS development or adjustments.

There are many approaches that furnish design guidelines for the major OS's parts, for example: task manager, memory manager, just to quote some of those parts. It is well known that there are even many source-code examples showing how to assembly those major parts. Although there are many guidelines showing how to design OSs, they

do not show how to modify an OS when, for example, the target hardware-platform is different than the original hardware-platform that it was designed for. This may be problematic, especially in (ES) development, because the source-code must be constantly modified in order to meet the new hardware's requirements.

Most OS modifications are based on a "trial and error" approach and, obviously, this is not the best way to deal with this problem. On the other hand, there are generic tools, which are applicable to generic source-code development, but they do not provide special functions to deal with OS complexity, regarding its deep hardware-attachment.

The greatest motivation for this research was the realization (after having designing many embedded systems) that there is no methodology for porting OS for embedded systems. Although the methods discussed in this paper have, as a target, embedded systems, those methods should also work for other classes of systems.

As far as we know, up to present there are no appropriate methods for OS porting in the literature, which is clear to us for the extremely reduced number of papers on the subject. Realizing that, the authors conceived an appropriate method to help on this task. This paper describes a complete method designed to do that job, as well as provides a practical contribution to the field, since it presents an application, called *O*perating *S*ystem *POR*t *T*ool – OSPORT, which was developed according the proposed method, in order to guide the designer in the process of porting an OS into a new hardware.

2. The Method which OSPORT Relies on

The OSPORT was developed based on a method that reveals all source-code files that must be analyzed, modified, reduced or increased in order to port or adjust an OS into a new hardware.

The method, in its turn, combines information obtained from the OS source-code and particularities of the new hardware-platform, and it is based on *source-code cross-reference*. The resulting cross-reference information shows all required source-code to provide software-support to the hardware.

A table holding the source-code relationship, called crossover-table, must be created when executing the cross-reference step. Creating such table requires expert knowledge about the OS and the target hardware in order to bind the software and the respective hardware it must provide support for.

2.1. Method Overview

Table 1 shows the steps of the proposed method. They are enumerated from 0 to 4.

Table 2 shows the transitions for these steps. Eventually, any of these steps can be re-run during the project execution.

Table 1. Five Steps of the Method

Step	Description	Step	Description
N0	Evaluate Requirements through Decision Making Support	N3	Identify all source-code relationships
N1	Identify Hardware elements without software support	N4	Identify the source-code precedence
N2	Identify all source-code to be modified		

Table 2. Transitions for the Proposed Method

Transition	Description
T0-0	Evaluate Requirements through decision making support
T0-1	Impact of hardware-elements over software elements
T1-0	Reevaluation of hardware-elements impact over the decision making support
T2-0	Impact of software-elements over project efforts, required for project accomplishment
T0-2	Reevaluation of software-elements impact over the decision making support
T2-1	Impact of software availability over hardware definition
T1-2	Impact of hardware availability over software definition
T2-3	Impact of new software-elements over the dependency table
T3-3	Recursive definition of software-dependencies
T3-4	Software-dependencies impact over precedence table actualization
T4-3	Precedence table impact over the dependency table – Creating precedence table

Figure 1 shows a graph representing all transitions and steps regarding Table 1 and

Table 2 combination. Steps N0, N1 and N2 are strongly connected and most critical issues are addressed in their transitions. A detailed description of all steps is presented in the next sections.

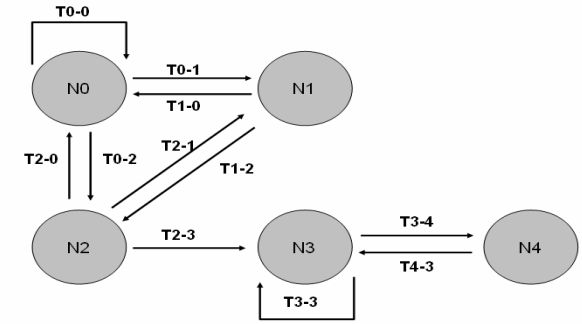

Figure 1. Five Steps Method Graph

2.1.1. Step N0 – Evaluate Requirements through Decision Making Support

The fundamental goal of step N0 is to submit the initial decision to the use of a Decision Making Model (DM) [MARKO 2001]. Figure 2 shows a suitable DM for an ES development project.

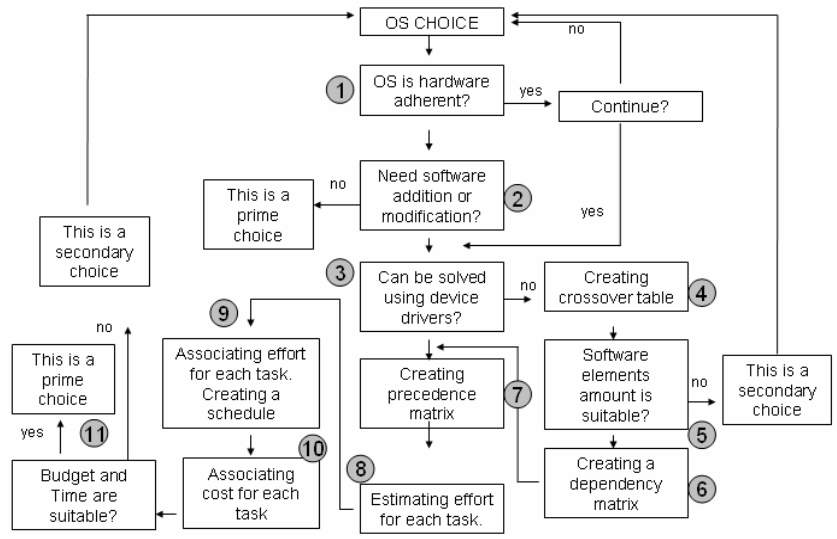

Figure 2. A Decision Making Model Applicable for ES

During project execution, DM can be adjusted in order to include new constraints. Of course, a suitable DM helps decreasing the effort on critical project's phases [MARKO 2001]. The DM recursively uses other steps from the proposed method in order to provide the following information: if the target OS is suitable for the target hardware-platform; all hardware-elements lacking OS support; and all software-elements needing modifications for the new hardware-platform.

2.1.2. Step N1 – Identify Hardware-Elements without Software Support

Step N1 aims at identifying discrepancies in the target hardware platform for the chosen OS. The crossover-table is the main result obtained after this step is done.

If a different hardware element is present in the new hardware-platform, then a similar[1] element must be selected. If the different hardware does not have any similar element supported in the chosen OS, the way to keep them working is by developing and using device drivers in order to provide OS supporting. The crossover-table's initial version is built by identifying the source-code and its respective hardware for what it provides support. By using this initial version it is possible to identify what hardware-element there is no software support.

2.1.3. Step N2 – Identify all source-code to be modified

The precise definition of all source-code to be modified requires the extension of the crossover-table, which takes place in steps N1 and N2. Please note that only hardware-dependent source-code must be considered. The table extension is obtained by applying the read-and-search approach over the source-code.

In order to identify all source-code needing modification, a cross-reference must be made. This cross-reference reveals source-code's relationship. If a source-code (*K*) requests services from other source-code (*L*) we named it: "source-code's relationship".

[1] Similar hardware means another hardware with the same functions

Referring to figure 3, it can be seen that, a modification on (*L)* reflects on (*K)*, and modifications on (*K)* also affects (*L)*. This figure also depicts a group of source-code's relationship which must be considered when modifying a source-code.

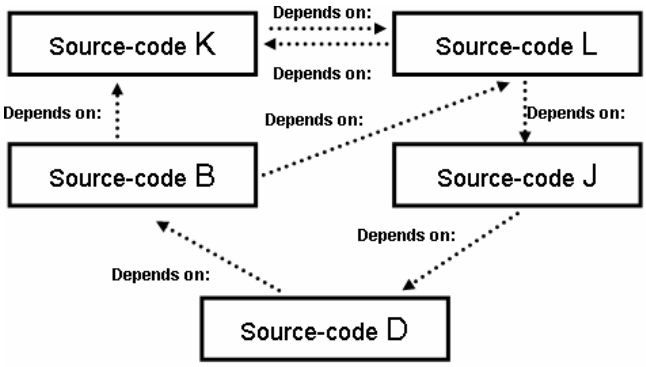

Figure 3. Dependency Matrix Template

2.1.4. Steps N3 and N4

Three kinds of essential pieces of information are revealed through the crossover-table: dependencies, requirements, and precedence.

Dependency identifies the interconnection's level of the source-code needing modifications. Dependency's level of a given source-code is obtained by adding all the other source-codes from which the given source-code depends on.

In general, the dependency computation is carried out as seen in (1) and the total of requirements as seen in (2).

$$Dep(x) = \sum_{1}^{k} i \qquad \begin{cases} i{=}1 \text{ if x depends of } k_i \text{ source-code} \\ i{=}0 \text{ if x does not depends} \end{cases} \qquad \begin{array}{l} \textit{k means the number of source-codes to search for} \\ \textit{x means the target source-code to get the dependency value} \end{array} \qquad (1)$$

$$\mathrm{Re}\,q(x) = \sum_{1}^{k} i \qquad \begin{cases} i{=}1 \text{ if x provides services for } k_i \text{ source-code} \\ i{=}0 \text{ if x does not provide} \end{cases} \qquad \begin{array}{l} \textit{k means the number of source-codes to search for} \\ \textit{x means the target source-code to get the requirement value} \end{array} \qquad (2)$$

The work needed for obtaining the information related to dependency and requirements of a source-code is a read-and-search job inside the OS source-codes.

Special care must be taken when solving cyclic references between source-codes since they could result in inadequate precedence classification. Such a problem occurs when a source-code file uses services implemented in another source-code.

Cyclic references must be resolved before the precedence between routines is computed. A strategy for solving this problem consists on using the Mock Object Pattern for creating temporary replacement routines [BROWN 2003].

In order to distribute the large information volume to be manipulated when porting an OS, a template/model for crossover-reference table called Dependency Matrix is proposed in the following subsection.

2.1.4.1. Step N3 – Creating a Dependency Matrix

Figure 3 depicts a dependency matrix, proposed for ES development. All source-code (subcomponents) and source-code families (components) that must be modified are placed in this matrix.

For the purpose of clearness, the example shown in Figure 4 only presents the subcomponent's level. Again, it is important to observe that only the source-code to be ported must be referenced.

The matrix is filled by marking a component that depends on other component. In order to do that, each subcomponent (in a row) is inspected by marking all cells in that row containing a subcomponent (in a column) from which the given subcomponent depends on.

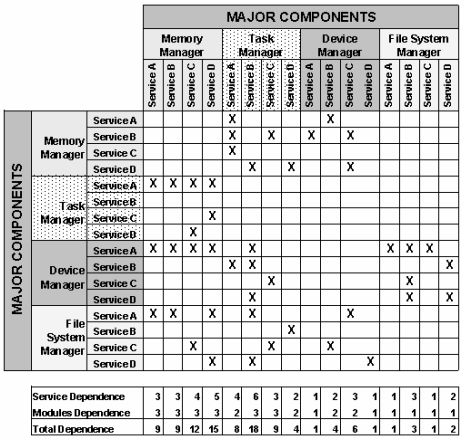

Figure 4. Dependency Matrix Template

2.1.4.2. Step N4 – Creating a Precedence Matrix

A precedence matrix is obtained from information resulting from the processing of the dependency matrix.

The Algorithm 1, part of the proposed method, classifies all routines held in the precedence matrix at the same time as it also solves cyclic reference issues. This processing considers the dependencies and requirements computed using (1) and (2), respectively. Once the *dep*(s) and *req*(s) values are defined, all information needed in order to provide data required in steps 1, 2 and 3 of Algorithm 1 is available. Figure 5 contains an example of a precedence matrix that is obtained after applying the algorithm.

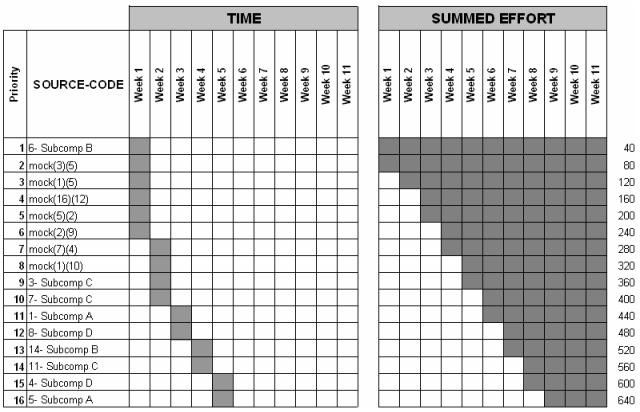

Figure 5. Precedence Matrix.

1. *For each routine, the totals of dependencies are registered in a list.*
2. *Let DEP(i)=the number of dependencies of the given routine*
3. *Let REQ(i)= the number of requirements of the given routine*
4. *Make SUP(i) = 0*
5. *Make PEND(i) = DEP(i)*
6. *Make PRIORI(i) = 0*
7. *Make PRIORITY = 1*
8. *The dependency list is classified in ascending order based on PEND and in descending order based on REQ*
9. *For each routine pair R(i) and R(k) com PEND(i)>0, and cyclic dependence::*
    1. *A mock routine named mock(i)(k) is created*
    2. *Make REQ(mock(i)(k))=1: Make DEP(mock(i)(k))=0: Make SUP(mock(i)(k))=0*
    3. *Add mock(i)(k) to the dependencies of R(i)*
    4. *Remove R(k) from the dependencies of R(i)*
10. *Classify the dependence list in ascending order based on PEND and descending order based on REQ*
11. *For each routine R(i) such as PEND(i) = 0 and PRIORI(i) = 0, do:*
    1. *Make PRIORI(i) = PRIORITY: PRIORITY = PRIORITY + 1*
    2. *For each routine R(k) depending on R(i), do:*
        1. *SUP(k)= SUP(k)+1*
        2. *PEND(k)= DEP(k) – SUP(k)*
12. *Repeat step 10 until all routines are prioritized*
13. *Classify the dependency list in ascending order by PRIORI*

Algorithm 1. Priority Computation and Cyclic Reference Removal.

3. The OSPORT Specification

The fundamental steps in the presented method were applied by designing an application dealing with Linux kernel source-code. Although method is applicable to any OS, it should be appreciated that, since Linux is an open-source code OS, more information that is necessary to apply the method is available than that provided by an analysis carried out on a non-open-source OS. On the other hand, it should be also stressed that open-source OS is being much more preferred in the project of embedded systems since one has not to pay royalties, which minimizes the total cost of ownership

(TCO). As a result, the authors wondered that the best flavor of the method would be better captured by the readers if an example regarding Linux was given.

OSPORT was developed according to JAVA 1.5 specifications and it runs under Linux or Windows. Table 3 depicts the specification for OSPORT application.

Table 3. OSPORT Specification

Feature	Specification
Development Platform	JAVA 1.5.0_08 / NetBeans IDE 5.5
Operating System Platform	Linux; Windows XP
Data Base Engine	MySQL 5.0
C Parser	Built in
Assembly Parser	Built in
Operating System Target	Linux Source-code

The following sections present details for OSPORT's implementation.

3.1. The Architecture

The OSPORT has a modular architecture, which makes it easier the process of adjusting it for other OS than Linux. Figure 6 depicts its functional architecture.

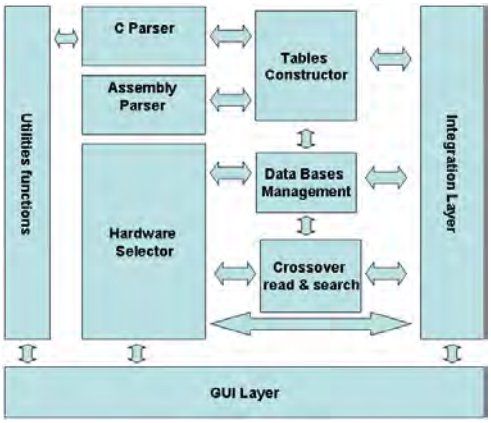

Figure 6. OSPORT's Layer Architecture

C and Assembly parsers are required in order to build the intermediate information which is useful for the read-and-search approach; this information is recorded into the MySQL data-bases.

The hardware selector uses the Linux configuration schema for gathering all relevant information for providing, in a basic and effortless way, the hardware versus software combination. That procedure minimizes the human-knowledge dependency but, if necessary, this combination can be extended by the user through OSPORT's functions. Mapping hardware versus software relationship helps identifying hardware elements without software-support, and it also helps deciding what source-code must be modified in order to match some hardware difference. The crossover module traces the source-code, recording all information that means relationships, and building a cross-

reference database. The table constructor builds all tables and matrixes from the relationships revealed by the crossover module.

3.2. Running OSPORT

The OSPORT was designed to be an user-friendly application, simple & easy. Example of some of its graphical interfaces is shown in Figures 7.

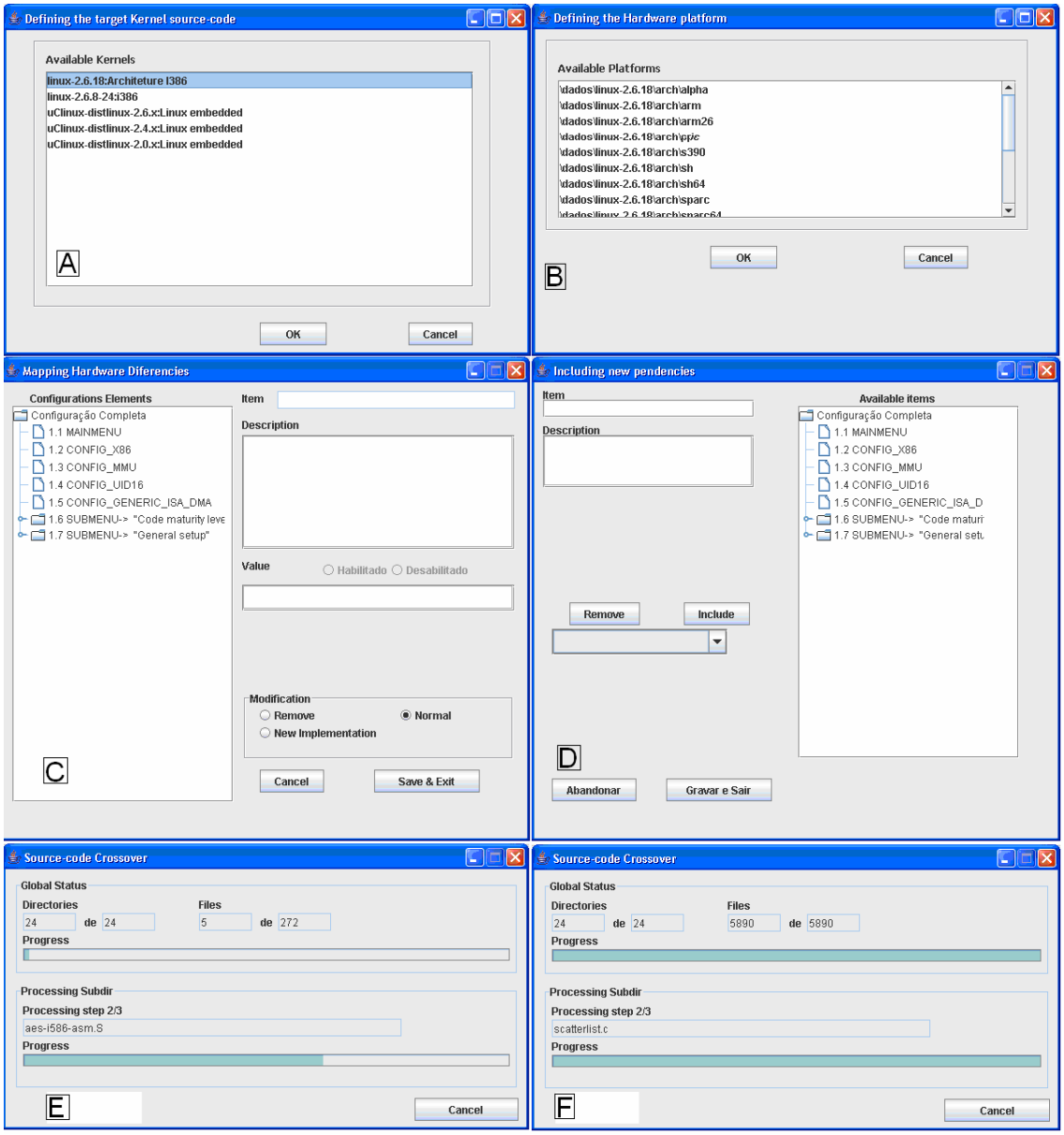

Figure 7. OSPORT's Graphical Interface

The steps the user should follow, when using the OSPORT application, are:

1- Select the Linux version using the *platform/Set Kernel* option (Figure 7 (A));

2- Select the similar hardware-platform for the target hardware-platform (Figure 7 (B));

3- Map all hardware differences from the similar hardware-platform concerning the target hardware-platform (Figure 7 (C));

4- Include new dependencies in order to deal with: non-similar hardware, which does not match any options in step 3; software-features modification; kernel-hacking modification (Figure 7 (D));

5- Start source-code cross-reference (Figure 7 (E));

6- After starting the source-code cross-reference (Figure 7 (F)), the user can use OSPORT to build the tables using the *Tables* option, in the main menu.

3. A Study Case

As a study case, we present the results obtained from OSPORT working with Linux 2.6.0 source-code.

In this non-complex study, the goal is to obtain a simple list of all source-code that must be analyzed in order to adjust some Linux-kernel features for an EBSA-110 similar board system. Another considerable study case, with OMAP5912 and OMAP5912 Starter Kit [OMAP 2007] (which is based on a dual-core processor that consists of a DSP and an ARM) is under development, in order to validate the OSPORT's accuracy in large OS-porting projects. EBSA-110 is an evaluation board for a StrongARM processor, available from Digital, and it is currently fully supported by Linux, which makes the job of doing kernel adjustments on the similar board, a fairly easy task. This board has limited on-board hardware features, including an onboard Ethernet interface, two PCMCIA sockets, two serial ports and a parallel port. The target hardware-platform, similar to EBSA-110, has no parallel ports, and it has one Bluetooth device.

3.2. Results

The cross-reference process only involved 8591 source-code files out of 18747 available Linux kernel files. The OSPORT's process takes about 20 minutes and it produces the results shown in Table 4, which contains the list of source-code matching the target system-board. Note that this list does not mean the source-code that MUST be modified. It shows the source-code that MUST be analyzed, and that probably will need modifications.

Source-code files presented in the element's list are related with the modifications required in order to adjust the OS for the target system-board, concerning its differences from the EBSA-110 system board.

Table 4. Source-code files list

Source-code file	Source-code file
include\linux\autoconf.h	include\linux\parport.h
include\linux\autoconf.h	drivers\bluetooth\bcm203x.c
drivers\bluetooth\bfusb.c	drivers\bluetooth\hci_usb.c
drivers\bluetooth\hci_bcsp.c	drivers\bluetooth\hci_ldisc.c
drivers\bluetooth\hci_h4.c	drivers\bluetooth\bpa10x.c
drivers\bluetooth\hci_vhci.c	drivers\bluetooth\hci_uart.h
drivers\bluetooth\hci_event.c	net\bluetooth\hci_conn.c
net\af_bluetooth.c	net\hci_core.c
net\l2cap.c	net\hci_sock.c
net\hci_sysfs.c	net\sco.c
net\bluetooth\rfcomm\tty.c	net\bluetooth\rfcomm\sock.c
net\bluetooth\rfcomm\core.c	net\bluetooth\bnep\sock.c
net\bluetooth\bnep\netdev.c	net\bluetooth\bnep\core.c
net\bluetooth\cmtp\capi.c	net\bluetooth\cmtp\sock.c
net\bluetooth\cmtp\core.c	net\bluetooth\hidp\sock.c
net\bluetooth\hidp\core.c	drivers\char\lp.c
drivers\parport\procfs.c	drivers\parport\ieee1284.c
drivers\parport\parport_pc.c	drivers\parport\share.c
drivers\parport\ieee1284_ops.c	drivers\parisc\superio.c
drivers\block\paridade\bpck6.c	drivers\block\paridade\paride.c

3. Conclusions

Developing systems without a well-defined approach may result in unstable or incomplete projects, regarding the OS. As it could be seen, the OSPORT application covers the essential steps for porting or adjusting an OS for a new hardware-platform.

The proposed approach for obtaining dependencies between source-codes comprising the OS assures that all adjustments are considered. The proposed method aggregates quality to the embedded-system development projects by providing a complete and anticipated view of the development and adjustment activities needing to be performed when porting an OS.

Most embedded-system projects and PC based projects deal with OS adjustment and it is important to count on applications that help the designer on the decision of what, and where, modifications should be done in the project. OSPORT application aims at providing efficient support by determining, at an early phase, all source-code that must be analyzed. As far as we know, this is an original work addressing this specific issue.

4. References

BOOCH, G., Object-oriented development. IEEE Transactions on Software Engineering. Vol. SE-12, no. 2, pp. 211-221., 1986

BOOCH, G, "On Architecture," IEEE Software, vol. 23, no. 2, pp. 16-18, Mar/Apr, 2006.

BROWN, M. A, TAPOLESANYI, E., Mock Object Patterns, Version 1.2.3 – 2003.

CAMPBELL, H. R., et al, CHOICES (Class Hierarchical Open Interface for Custom Embedded Systems), Operating Systems Review, 21(3):9-17, 1987.

CARRO, L.; WAGNER, R. F., Sistemas Computacionais Embarcados, 2º capítulo, 2004.

FROHICH, A. A. M, Application-Oriented Operating Systems, Dissertação de mestrado Universidade Federal de Santa Catarina, 2001

LEE E.A., What's Ahead form Embedded Software? - IEEE Computer, September 2000.

PARNAS, D. L., On the Design and Development of Program Families, IEEE Vol SE-2 Nº 1, 1976.

POSADAS, H., et al, Single Source Design Environment for Embedded Systems Based on SystemC, Design Automation for Embedded System, 9, 293-312, 2004.

POLPETA, F. V., FROHICH. A. A. M, Um Método para a Geração de Sistemas Embutidos Orientados a Aplicação Baseados em SoCs, XXV Congresso SBC 3129-2005.

MARKO BOHANEC. IN C. BAVEC et al., editor, Proceedings of the 4th International Multi-conference Information Society 2001, volume A, pages 86--89, Ljubljana, October 2001.

OMAP, The OMAP5912 Starter Kit (OSK), website address: http://www.kanecomputing.co.uk/osk_omap5912.htm, 2007.