2013 IEEE 37th Annual Computer Software and Applications Conference Workshops

Partitioning Decision Process for Embedded Hardware and Software Deployment

Gaetana Sapienza, Tiberiu Seceleanu
ABB Corporate Research
and Mälardalen University,
School of Innovation, Design and Engineering
Västerås, Sweden
{gaetana.sapienza, tiberiu.seceleanu}@se.abb.com

Ivica Crnknovic
Mälardalen University,
School of Innovation, Design and Engineering
Västerås, Sweden
ivica.crnkovic@mdh.se

Abstract— **Many types of embedded systems applications are implemented as a combination of software and hardware. For such systems the mapping of the application units into hardware and software, i.e. the partitioning process, is a key phase of the design. Although there exist techniques for partitioning, the entire process, in particular in relation to different application requirements and project constraints, is not properly supported. This leads to several unplanned iterations, redesigns and interruptions due to uncontrolled dependencies between hardware and software parts. In order to overcome these problems, we provide a design process that enables the partitioning based on a multiple criteria decision analysis in a late design phase. We illustrate the proposed approach and provide a proof-of concept on an industrial case study to validate the approach applicability.**

Keywords—Development Process, Model-based Design, Component-based System, Partitioning Decision Process, Multiple Criteria Decision Analysis (MCDA).

I. INTRODUCTION

In many embedded systems applications are implemented in hardware and in software. In this type of applications, a proper mapping on hardware and software units is very important, due to reasons such as performance, reliability, and costs. A suitable mapping poses strong requirements on design methods, in terms of effectiveness and efficiency.

Today, it is a rather common practice that at early stages the design is split into separated flows: hardware and software. As a consequence, the partitioning decision process - i.e. the process dealing with the decisions upon which parts of the application have to be designed in hardware and which in software - is not supported by any well-structured methodology. This leads to a number of issues (e.g. design flow interruptions, redesigns, undesired iterations, etc.) which negatively impacts the overall development process, the quality and lifecycle of the final system. Detailed problem statements related to the partitioning can be found in [1], [23].

Starting from the 1990s, an intensive research work was performed, focusing on partitioning techniques which tackled solutions satisfying mainly low-level performance and resource utilization requirements [2], and several partitioning approaches were proposed [3][10]. During the last years, the importance of a well-defined and effective partitioning decision process is obfuscated by tools and integrated co-design environments (e.g. MathWorks Simulink® [19], Space Codesign® Systems SpaceStudio™ [20]) which well-support approaches such as "trial and error".

The increasing complexity of the applications is also leading to an increased architecture complexity and to a large number of components and communications between them. This has impact on the partition which process becomes more intricate, and more difficult to manually obtain good results. In addition to this, many project constraints, such as cost reduction, short lead time, have impact on the partition process since different efforts are for different implementation. Finally the non-functional requirements such as safety, reliability, and run-time resource constraints have impact on the partition decision. This all makes the partition process complex, dependent on many variables and this lead to strong needs for an efficient and automated partition process that provides an acceptable solution in the given conditions.

In this paper we are proposing a new partitioning method that comprises a complete development process from the requirements management, architectural design, component modeling, to the decision for their implementation either as software or hardware components. Our contribution in this paper can be summarized as follows. First we present a new systematic partitioning methodology (i) enabling technology-independent design in an early stage of the design and reusing existing solutions, i.e. functional units implemented either as software or hardware; (ii) performing the partitioning in a late stage of the design based on a multiple and even conflicting set of criteria derived by the overall application requirements, system constraints (such as memory capacity, or process power), and the project constraints (such as efforts, costs, or time). Secondly, we establish a tool chain for supporting the methodology. Lastly, to demonstrate the viability of the approach, we provide a proof-of-concept on an industrial case study.

The rest of the paper is organized as follows. Section 2 defines the main problem and states the main objective. Section 3 presents the new proposed methodology. Section 4 illustrates the industrial case study. Section 5 presents the related work, and finally Section 6 concludes the paper and discusses future work.

978-0-7695-4987-3/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSACW.2013.131

II. Research Problem and Objective

For embedded systems built on heterogeneous platforms (e.g. a platform consisting of diverse computational units, for instance, an Field Programmable Gate Array (FPGA), microprocessor and graphics processing unit GPU), a specific activity of the design phase is to decide about the application deployment. Assuming that an application is implemented by a set of interacting components, the deployment decision is transformed to a setoff decisions for each component, whether a component will be implemented as software (e.g. C/C++ code) or as hardware (e.g. VHDL, etc.).

It is common practice that deployment decisions are taken at an early stage of the design phase, and that it branches into two separated flows: hardware and software design flows. Then, they evolve separately until the final integration during the implementation phase. Figure 1 shows a simplified diagram of a traditional development process. It is not rare that the phases get interleaved and that iterations and/or optimizations are needed. In this scenario, the design phase is affected by issues such as hardware or software flow interruptions (due their mutual dependencies), redesigns and unplanned iterations which negatively impact the overall development process in terms of efficiency, quality and costs, and the system lifecycle. Although hardware and software for embedded applications are tightly connected: the (i) hardware design does not take into account the computational power required by software and the capability that the software might offer for enabling hardware optimization and (ii) software design does not impact the hardware design specifications, and does not fully exploit the available hardware resources. Beside this, the separation into software and hardware often occurs without the support of an accurate and well-structured partitioning decision process. Decisions are not the results of an accurate trade-off analysis taking into account the large and even conflicting number of requirements and project constraints that nowadays - and even more in the future - are required to develop complex and sustainable applications.

To overcome these problems, our main research objective is to provide a methodology for enabling technology-independent design and pushing partitioning decisions to a late stage. Further, the partitioning decisions should be the results of many requirements and constraints, which in our method is achieved through a Multiple Criteria Decision Analysis (MCDA).

To precisely specify the results we have defined the following research questions.

- *How to properly enable technology-independent design in the earlier stage of the design phase and perform the partitioning decision process in a later stage?*
- *How to enable a systematic and effective process that supports the design engineers before partitioning?*
- *How to provide an effective and accurate partitioning decision process providing optimal and sustainable results which taken into account requirements and project constraints?*

III. The Partitioning Decision Process

In order to address to the aforementioned questions, we propose and design a systematic decision process for partitioning the application into hardware and software. It allows common model-based design first and enables the separation into hardware-specific design and software-specific design in late stage.

Foundations. Our approach is inspired by Model-Driven Architecture with Platform-Independent Model (PIM) and Platform-Specific Model (PSM) stages [4] and supported by Model-based [21] and Component-based approaches [22]. The latter is a well-known approach in software development but it not used for development of both hardware and software, so specifically we extend the approach to hardware components as well. Consequently, we model the application as set of components. In the PIM stage, the representation of the components is technology-independent. After the partitioning, which corresponds to the PSM stage, they are hardware-specific and software-

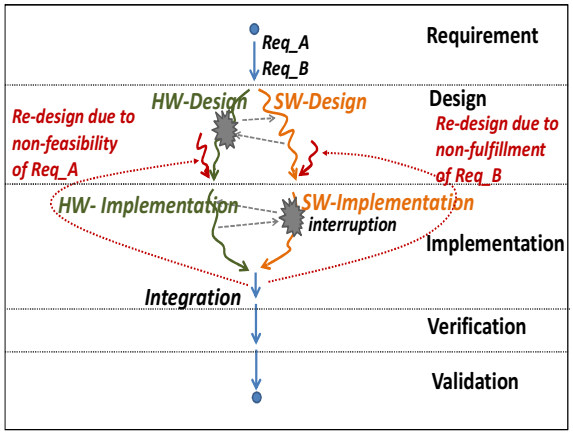

Figure 1- Traditional Development Process (Simplified Overview)

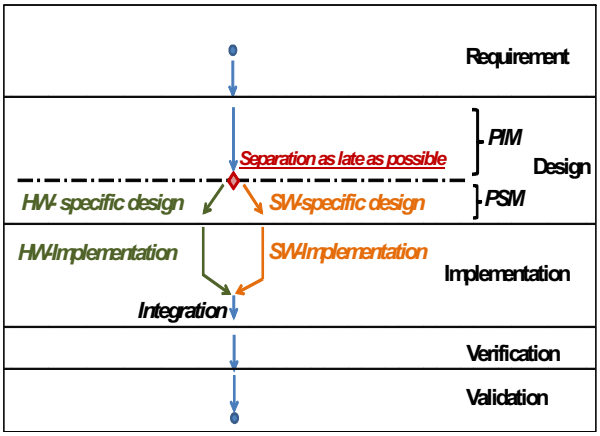

Figure 2 - New Proposed Approach

specific designed and implemented.

In order to formally provide a definition of application, hardware and software components, and their interconnections, we adapt and extend to hardware, the component definition given in [5]. Thus, the embedded systems can be seen as a component-based system (CBS) represented by a tern of elements:

$$CBS = \langle C,B,P \rangle$$

where (C) is the set of components representing the application, specifically they can be hardware or software; (B) the set of bindings between the components; (P) the platform on which the components are deployed. This latter is already given as result of project constraints.

The CBS is derived by application requirements and project constraints. Figure 3 provides a diagram of the CBS.

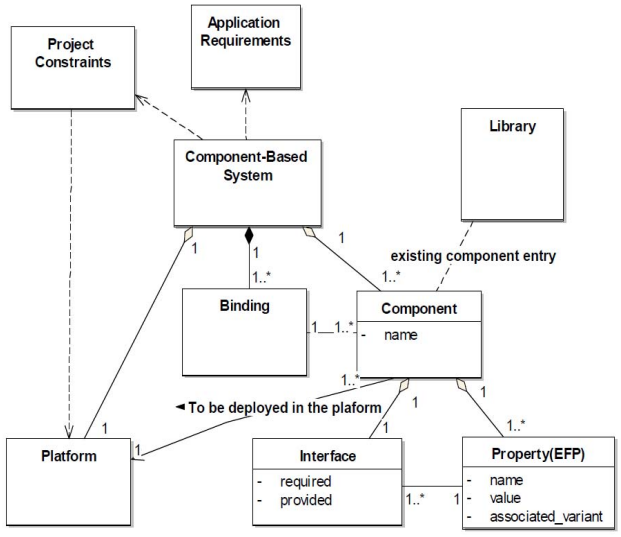

Figure 3 - Component-based Systems Diagram. Component Library

By extending the definition provided in [5] to hardware components as well, a component ***(C)*** is defined by:

$$C = \{I,P\}$$

- an interface ***(I)*** which characterizes the component from a functional perspective. The interface consists of two parts: required and provided
- a set of properties ***(P)***, which specify the a non-functional perspective. These properties also called extra-functional properties (EFP).

An example of component is given by a PI-controller. Its interface is represented by the required signals (set-point and feedback) and the provided signal (regulated output). Execution time, accuracy, energy consumption, reliability are some instances of EFP for this component.

We expect that a component can have different implementations (hardware or software), which we refer as variants. For each component, the interface remains the same, but the set of properties *(P)* or the properties values are different for each variant. For example, the value of the worst-case execution time is different between a hardware variant and a software one.

High-level Reuse. In addition to this, the approach enables high-level component reuse. In order to achieve this, a component library is built. It includes existing components and their variants.

Each entry (i.e. a component) in the library consists of information about the interface and the EFP associated to each variant. The properties include characteristics of the component themselves, and specifications of the execution context, such as the type of platform in which the variant runs on. In Figure 4, an example of entry is highlighted. It is a Power Regulator component (C2). Its interface consists of two required signals and two provided signals. It has hardware and software variants, and each variant has a number of EFP values associated, as shown in Figure 4.

Process Activities and Partitioning Decision Table Building. Here, we briefly describe the main activities of the designed methodology for enabling a systematic partitioning process. Along with the activities, a key artifact, called partitioning decision table is built. Components and their properties are the basic information included in this table. Based on it, partitioning decisions are evaluated and taken. As a consequence, it is of crucial importance to ensure that the table is properly built and that it contains all the relevant information needed to perform a successful partitioning. An example of the table is provided in Figure 4. The main activities are listed below.

a) Modeling of the application as a set of components. The application is modeled as a number of interconnected components. The modeling is carried out based on the application requirements and the information available in the library. This latter provides, to the designers, a mean to take into account previous expertise, to give feedback to the requirements engineers in case of requirements incompleteness and to speed-up the modeling activities by component reuse. At the end of this activity, the application architecture (i.e. components and bindings) is defined. Each component has an entry in the partitioning decision table as shown Figure 4. Each component is identified, with respect to the library, as existing one (belonging to "Set A") and non-existing one (belonging to "Set B"). Set A and Set B are shown in Figure 4. For existing components EFP information are retrieved from the library. For new ones, two possible variants are associated: hardware and software and the related EFP values are estimated.

b) Identification of overall application and project constraints to derive decision criteria. Based on overall application or project constraints a number of system decision criteria are identified. These criteria address the overall or part of the architecture. For instance, we can have criteria derived by a deployment constraint: two components have to be deployed as software.

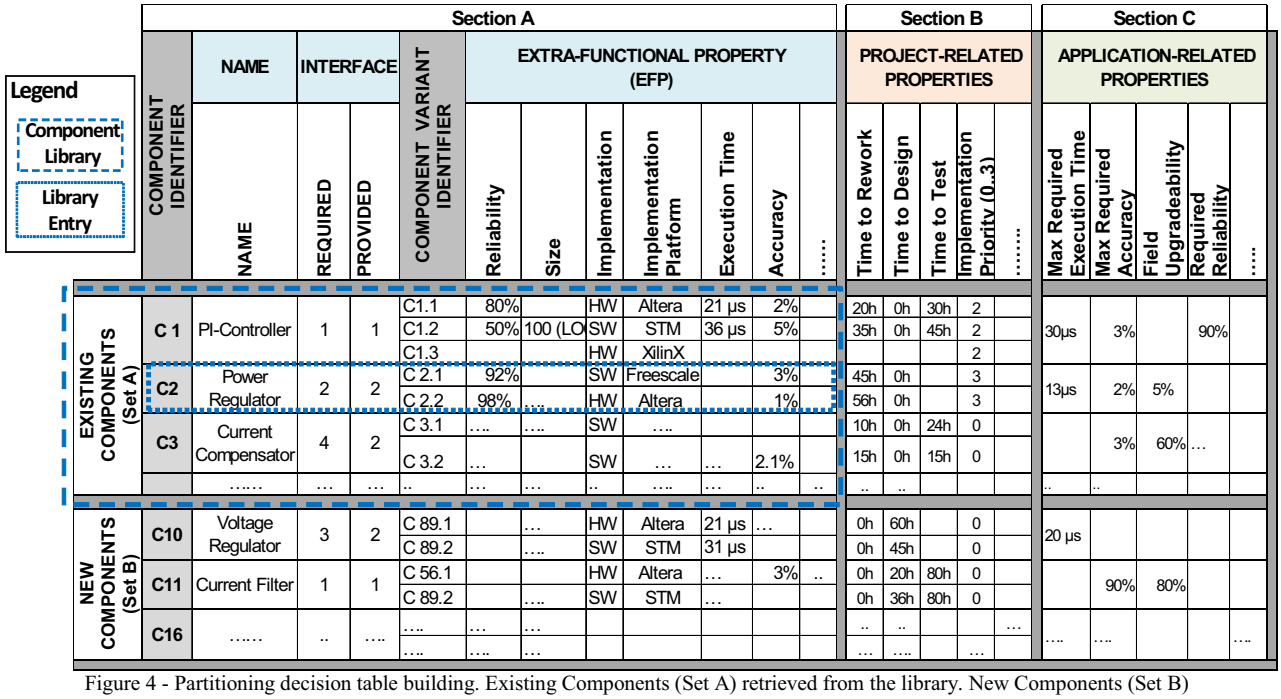

Figure 4 - Partitioning decision table building. Existing Components (Set A) retrieved from the library. New Components (Set B)

c) **Identification of project- and application-related properties.** The identification of component properties derived from (i) project constraints and (ii) application constraints is carried out. The identified properties are respectively added into the partitioning decision table. Examples of these properties are provided in Figure 4. Section B contains the properties related to project constraints. Section C includes the one related to application constraints. Subsequently, a value is assigned to each variant.

d) **Filtering and Property prioritization.** All variants which do not satisfy application and project constraints are filtered out. The next step is to assign a priority to the most relevant properties. It is carried out by assigning weights.

e) **Component variants selection.** Performing the partitioning of component variants based on (i) the criteria defined at point b, (ii) the properties values assigned at point c and (iii) the property weights decided at point e. The expected outcome is either a single partitioning solution or several ones. It is achieved by applying MCDA methods for selecting the component variants.

f) **Solution ranking.** In case of multiple solutions, further decision criteria need to be defined by the design engineers. Based on these criteria, suitable MCDA methods for ranking the solutions will be applied.

In case the partitioning process does not converge to any feasible solution, the process is reviewed and new iterations are performed.

IV. THE INDUSTRIAL CASE STUDY

In order to validate the research work, we follow the guidelines proposed by Shaw in [7]. We base our validation strategy on a question related to the feasibility of the proposed overall process: *Is this defined process feasible, viable?*

For this purpose, we applied the methodology on a real industrial application, developed for the Artemisia iFEST (industrial Framework for Embedded Systems Tools) project [7]. Here, a wind turbine control application is designed and deployed as a prototype. A wind turbine converts the rotational mechanical energy of the rotor blades into electrical energy, which will be distributed further via a power network. The core element of our application is the controller, which has to dynamically regulate the rotor blades' pitch at different wind profiles while maximizing the generation of electrical energy. In the project, the application partitioning is carried out without the support of a systematic partitioning methodology. Partitioning decisions are performed in a relative early stage of the design phase. They are mostly based on the software and hardware designer expertise and they are also not pondered with respect to any project constraint.

Our main idea is to show the viability of our partitioning decision process by using the same application. Initial steps on this direction were presented in [6]. Here, we complement the existing iFEST methodology and tool chain with MCDA techniques and tools. The list of the main used tools is given as follows.

- HP – ALM (Hewlett-Packard Application Lifecycle management) [18]: for specifying and analyzing the requirements.

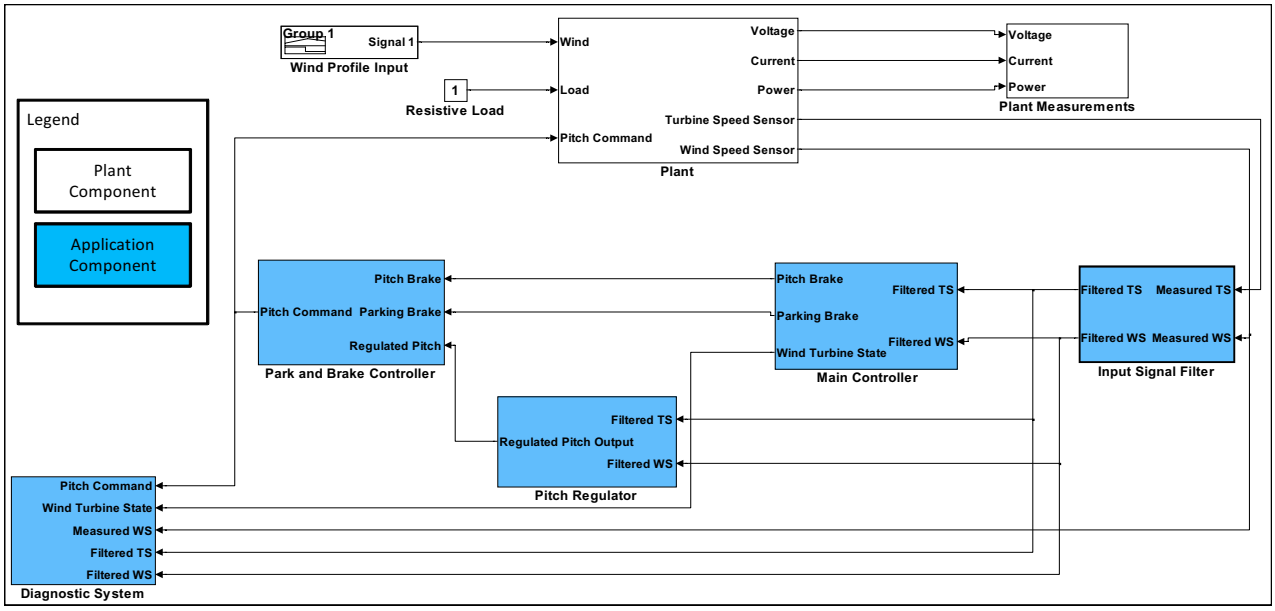

Figure 5 - Wind Turbine Application and Plant Model – Component Model (Simulink).

- MathWorks Simulink [19]: mostly for the design and implementation (automatic generation of C-code and VHDL) but also for the verification and validation of the application.
- System for ANalysis of Alternatives (SANNA) [17]: a spreadsheet-based tool for solving MCDA problems.

We start by specifying the requirements from a functional and extra-functional perspective. They are subsequently provided as input for (i) the application modeling and for the component selection; (ii) the extra-functional properties identification of the components; (iii) the identification of project and application constraint properties.

We continue by modeling the application as a number of interconnected components. The final architecture is shown in Figure 5, through a Simulink model. Each box represents a component. The core functionalities are modeled by: the main controller (*Main Controller*) which directs the overall control; the pitch regulator (*Pitch Regulator*) which calculates the pitch angle; and (iii) the park and brake controller (*Park and Brake Controller*) which is responsible for the park or brake of the turbine. In addition to this, it is required to transduce and filter the input signals to the main controller (*Input Signal Filter*), and to have a diagnostic system of the turbine (*Diagnostic System*).

Besides taking into account the application requirements, the modeling activity is supported by taking into account the project constraints and the component library information. Example of project constraint is the platform on which to deploy the application, i.e. a combined technology solution of FPGA and CPU (belonging to the Xilinx Zynq-7000 product family). This implies that each component has to be deployed either on the FPGA (hardware) or on the CPU (software).

At the end of this activity, the component interface is identified in terms of the required and provided parts for each component, as it can be seen in each component model in Figure 5 and Figure 6. For instance, the *Main Controller* component consists of the *Filtered TS* and *Filtered WS* as a required part and of *Pitch Brake, Parking Brake* and *Wind Turbine State* as provided one. In addition to this, all components are classified as existing (Set A) or new (Set B) ones with respect to the library, as shown in Figure 6. For instance, for the *Main Controller* component there are two existing software variants in the library, while for the *Diagnostic System* component two new virtual variants (hardware and software) are associated and inserted as entry in the partitioning decision table. As next step, the EFP of interest for the decision partitioning process, based on engineer expertise are identified and estimated. Few examples are: the execution time, the component size, the reliability, etc. In Figure 6, a simplified version of the partitioning decision table is shown. An example of project–related and application-related properties is shown as well.

Subsequently, we perform filtering operations to remove the variants which do not satisfy the application or project constraints. In Figure 6, just the relevant variants are shown. After that, we assign weight values to the properties, in order to prioritize these latter. Initially it is assumed that all properties are equally important. Consequently, the same weight value is assigned to the two most important properties that we consider here: the execution time and development effort (Figure 6). The weight values are normalized.

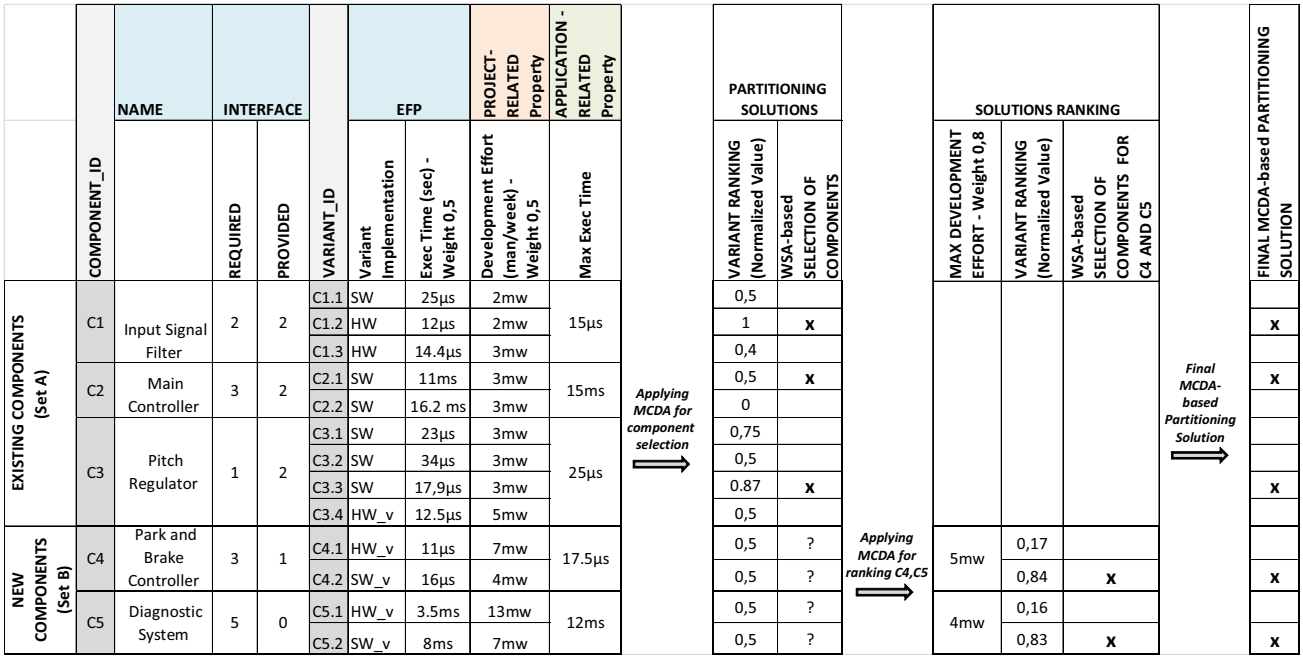

Figure 6 - Simplified Partitioning Decision Table (left). MCDA-based Partitioning Process. Final Partitioning Solution

Based on the information available in the table (i.e. component variants and properties values) and the properties priority, we have used the MCDA-based spreadsheet to select the components. Specifically, for performing the selection we use the Weighting Sum Approach or Model (WSA or WSM) method [17]. This method computes a global performance index related to each alternative (i.e. component variants) through the normalized weighted sum of each criterion. For C1, C2, and C3 a variant is selected, as shown by Figure 6, in the *WSA-based SELECTION OF COMPONENTS* column. The selection is performed based on the value of the *VARIANT RANKING (Normalized Value)* parameter (Figure 6), which is the performance index associated to each alternative computed through the WSA-method. This parameter has the same value for both C4 and C5 variants. Hence, several feasible partitioning solutions are available. In order to decide which solution to adopt, an additional decision criterion is defined which is derived by project constraints: the maximum acceptable development effort *(MAX DEVELOPMENT EFFORT)* for C4 and C5. Based on this, we assign new weight values to the properties and WSA-based calculations are iterated for C4 and C5. As a consequence, new ranking values for C4 and C5 are obtained, as shown in Figure 6.

The final partitioning solution is reached as follows: C2, C3, C4 and C5 are deployed as software (on the CPU) while C1 as hardware (on the FPGA), as shown in Figure 6, through the *FINAL MCDA-based PARTITIONING SOLUTION* column. With respect to the library, C1, C2 and C3 are reused, even though a virtual variant is taken into account for C3.

For each activity (see Section III), we performed analysis and verification.

In order to validate the design, we simulate the application using a model of a plant. This is calibrated against the turbine prototype.

V. Related Work

Partitioning of application into hardware and software is considered to be a NP (non-deterministic polynomial)-Hard problem [9]. It is an extensively studied topic; classical approaches based on heuristic, iterative and clustering algorithms are presented and discussed in [10], [24],[25]. A sophisticated integer linear programming model for joint partitioning and scheduling is presented in [11]. Additional approaches like Genetic Algorithm and Artificial Neural Network are proposed in [12] and [13]. However, such approaches are mainly focused to address one specific criteria, e.g., component execution time, component power or memory consumption. The approach proposed in [14] describes a scheme for achieving partitioning results targeting low power consumption and short execution time.

On the other hand, partitioning decisions, today, must account for application and project requirements as well as constraints, which move the focus of partitioning problem into a multi criteria perspective. Examples of work in this direction are provided by [15] where a MCDA approach is used for ranking different partitioning solutions based on trade-off analysis. A partitioning process able of supporting application constraints imposed by the reuse of existing modules in the automotive industry is presented in [16]. However, approaches that generate partitioning

solutions based on MCDA which takes into account both project and application properties are inexistent.

VI. CONCLUSIONS

The main outcome of this research work is the design of an overall process suitable for enabling (i) platform-independent design and reuse, and (ii) a systematic decision process to partition applications in a late stage of the design phase. More in details the contribution includes:

- The definition of a component model that suites well both software and hardware components;
- The formalization of a systematic partitioning decision process, which in comparison with traditional approaches enables decisions accounting project and application constraints as well.
- The establishment of a tool chain for supporting the partitioning decision process, based on a MCDA approach.

In addition, we have shown the feasibility of the process via the development of an industrial application prototype.

Future Work. From the overall methodology definition, we see the need for the formalization of a meta-model for enabling an accurate modeling of hardware and software components. In addition to this, it is also relevant to perform a systematic review of MCDA-techniques and tools in order to identify more suitable and versatile ones.

ACKNOWLEDGMENT

This research is supported by the Knowledge Foundation through ITS-EASY, an Industrial Research School in Embedded Software and Systems, and by Swedish Foundation for Strategic Research through the RALF3 project, both affiliated with Mälardalen University, Sweden, and the ARTEMIS iFEST project.

REFERENCES

[1] G.De Micheli, R.Gupta, "Hardware/Software Co-Design," of the IEEE, vol. 85, No.3, pp.349-365, 1997.

[2] W.Wolf, "A Decade of Hardware/Software Codesign," IEEE Computer, vol. 36, no. 4, pp. 35 – 43, Apr. 2003.

[3] "Hardware/Software Codesign: The Past, the Present, and Predicting the Future", Proceedings of the IEEE, vol. 100, issue: Special Centennial Issue, pp. 1411-1430, May 2012.

[4] A. G. Kleppe, J. Warmer, W. Bast, "MDA Explained: The Model Driven Architecture: Practice and Promise", Addison-Wesley Professional, 1 edition, May 1 2003.

[5] I. Crnkovic, S. Sentilles, A. Vulgarakis, M.R.V. Chaudron. "A Classification Framework for Software Component Models", Software Engineering, IEEE Transactions on, vol.37, no.5, pp.593-615, Sept.-Oct. 2011.

[6] G.Sapienza, T.Seceleanu, I.Crnkovic. "Toward a methodology for hardware and software design separation in embedded systems", the Seventh International Conference on Software Engineering Advances, Nov. 2012.

[7] M. Shaw, "Writing Good Software Engineering Research Papers". In Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, pp. 726-736, 2003.

[8] iFEST - industrial Framework for Embedded Systems Tools. ARTEMIS JU project #100203. Retrieved October 31, 2012, from http://www.artemisifest.

[9] P. Arato, S. Juhasz, Z.A. Mann, A. Orban, D. Papp, "Hardware-software partitioning in embedded system design", Intelligent Signal Processing, IEEE International, pp. 197 – 202, 2003.

[10] M. L. Vallejo, J. C. López, C. Real, "On the hardware-software partitioning problem: System modeling and partitioning techniques", Journal ACM Transactions on Design Automation of Electronic Systems, Volume 8 Issue 3, pp. 269 – 297, July 2003.

[11] R. Niemann, P. Mawedel, "An algorithm for hardware/software partitioning using mixed integer linear programming", Design Automation for Embedded Systems, special issue: Partitioning Methods for Embedded Systems , vol. 2, pp. 165-193, March 1997.

[12] K. B. Chehida, M. Auguin, "HW/SW partitioning approach for reconfigurable system design", Proceedings Int. Conf. Compilers Arch. Synth. Embedded Syst., pp. 247–251, 2002.

[13] M.A. Dias, W.S. Lacerda, "Hardware/Software co-design using artificial neural network and evolutionary computing", 5th Southern Conference on Programmable Logic, April 2009.

[14] Y.Fan, T.Lee, "Grey Relational Hardware-Software Partitioning for Embedded Multiprocessor FPGA Systems", AISS: Advances in Information Sciences and Service Sciences, vol. 3, No. 3, pp. 32 – 39, 2011.

[15] P. Garg, A. Gupta, J.W. Rozenblit, "Performance analysis of embedded systems in the virtual component co-design environment", Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 61-68, May 2004.

[16] Baumgartner, W. Gauert, "Linking codesign and reuse in embedded systems design", Proceedings of the Eighth International Workshop on Hardware/Software Codesign, pp. 93-97, May 2000.

[17] Josef Jablonský. System for ANalysis of Alternatives, March 01, 2010. http://nb.vse.cz/~jablon/sanna.htm. Latest access, April 10, 2013.

[18] Application Lifecycle Management (ALM) | HP® Official Site. http://www8.hp.com/us/en/software-solutions/software.html?compURI=1215990.Latest access, April 10, 2013.

[19] SIMULINK, Simulation and Model-Based Design Official Site. http://www.mathworks.se/products/simulink/index.html. Latest access, April 10, 2013.

[20] Space Codesign® Systems SpaceStudio™ Official Site. http://www.spacecodesign.com/. Latest access, March 23, 2013.

[21] F. Paterno , Model-Based Design and Evaluation of Interactive Application. Springer Verlag, 1999.

[22] I. Crnkovic. Component-based software engineering for embedded systems. In Proceedings 27th International Conference on Software Engineering (ICSE2005), pages 712–713, Missouri, USA, 15-21 May 2005.

[23] S. Edward, L. Lavagno, E. Lee, and A. Sangiovanni, "Design of embedded systems: Formal models, validation and synthesis" Proc. IEEE, this issue, pp. 366–390.

[24] W. Ahmed and D. Myers. Concept-based partitioning for large multidomain multifunctional embedded systems. ACM Transactions on Design Automation of Electronic Systems, pp. 221- 229, 2010.

[25] J. Wu, Q. Sun, T. Srikanthan "Algorithmic aspects for multiple-choice hardware/software partitioning". Computers & Operations Research, Volume 39, Issue 12, pp. 3281–3292, Dec 2012.