Modelling for Hardware and Software Partitioning based on Multiple Properties

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

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

***Abstract*—In many embedded systems types the separation process for deploying the applications as software and hardware executable units, called partitioning is crucial. This is due to the fact that partitioning decisions impact the overall life cycle of the systems. In industry it is common practice to take partitioning decisions in an early stage of the design, based on hardware and software designers expertise. We propose a new methodology as a combination of model-based and component-based approaches which enables a late partitioning decisions based on high level system requirements and project constrains. The final partitioning is decided based on a multi-property analysis approach. Here, we focus on the formalization of the overall process and in particular on the definition of a comprehensive system metamodel. This is meant to support modelling approaches suitable for enabling both the partitioning and reuse. An industrial case study is used to illustrate the approach.**

***Keywords*-Application Development Process; Component-Based System (CBS); Component Properties; Partitioning; Multiple Criteria Decision Analysis (MCDA).**

I. Introduction

Many modern embedded systems are implemented as software (SW) and hardware (HW) components, i.e. deployed as SW on CPU platforms and as HW programmed in Field-Programmable Gate Array (FPGA). The heterogeneous platform dramatically increases the system performance. However, the increasing complexity of industrial embedded applications constantly challenge designers when making decisions upon the separation into HW and SW for their deployment. These design decisions are of crucial importance since they impact (i) the application performance and its quality, (ii) the entire development process, and (iii) system lifecycle management. They require to be taken upon several criteria which are derived from a number of requirements and project constraints, which differ, vary and have even conflicting priorities. Like in other architectural decisions, the partitioning (i.e. the SW/HW separation process) requires a trade-off analysis. Over the decades, several techniques and approaches have been proposed for SW/HW partitioning, but they were mostly driven by the technological advances in electronics performance [13],[8],[14] and often tackled the optimisation problem with focus on performance and costs [7], [8]. In industry, mainly due to time pressure imposed by the time-to-market and cost optimization, it is a common practice to take partitioning decisions in an early stage of the design phase and without the support of systematic approach. They are most likely based just upon HW and SW team expertise, which often is not synergetically combined [13]. The decisions are lacking a support obtained by a comprehensive exploration of different options with respect to requirements and project constraints. Overdesigns, underdesigns, redesigns, flow interruptions in design and implementation, unplanned iterations are some of the typical drawbacks generated by this practice [10].

This paper addresses these needs by introducing a new approach *Multiple Criteria-based Partitioning Methodology*, we called MultiPar.

This approach aims for (i) providing partitioning decision in a late stage of system design, and (ii) base the partitioning decision on many and diverse criteria. The main contributions in this paper are a) formal definition of a comprehensive metamodel able of describing both SW and HW units and b) description of the MultiPar process flow which is based on the following strategy: (i) enabling application technology-independent design in the early stage of the design phase and pushing the partitioning decisions for enabling platform-specific design to a late stage, (ii) enabling the application deployment into SW and HW based on multiple criteria decisions which account for the high level systems requirements and project constraints, and (iii) making also decisions based on a reusable set of alternatives, in order to enable design and implementation reusability.

We illustrate a model conforms to the metamodel and the basic concepts of MultiPar on an industrial case: a simple wind turbine control system. The rest of the paper is organized as follows: Section 2 presents the metamodel. Section 3 defines the MultiPar process, Section 4 illustrates a case study, Section 5 presents the related work and Section 6 concludes the paper and provides the future directions of this research work.

II. The Metamodel for Partitioning

Component models and metamodels. There exists several component models that are specified by means of metamodels, for example Pecos [9], and ProCom [12] which

978-0-7695-5091-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SEAA.2013.56

in a similar way define interfaces and Extra-Functional Properties (EFPs). Our approach extends these specifications by allowing variable management of EFPs with respect to different component variants. By allowing a variable set of EFPs for component variants we have made it possible to reason about very different implementations, like software and hardware, and we have made it possible to easier compare the exhibited properties with those that are required.

In order to formally and unambiguously capture the essential domain concepts for components that have to be partitioned into SW or HW and the relevant relationships among them we have developed a metamodel. Figure 1 shows a simplified overview of the diagram describing the proposed Component-based System (CBS) metamodel. It is also used along the following subsections to explain the key concepts of the metamodel. Each CBS is supposed to be identified, described and documented as shown in Figure 1 by the *Entity Identification* entity.

Component-Based System. The central core is represented by the *ComponentBasedSystem* entity which compliantly to the definition provided in [11]. It is composed by a platform, a set of both components and connectors.

Platform. The *Platform* entity is meant to abstract the model of an underlaying and already defined platform, on which an application is deployed in form of SW and HW components. It consists of PlatformComponent and PlatformConnector entities.

A *PlatformComponent* entity models either a digital or analog HW component or a low-level SW component. The *PlatformComponentType* entity is used to indicate if the PlatformComponent is an HW or SW. The application components are deployed on platform components. This is defined by the semantic relationship between the PlatformComponent and the Component entities named "*to be deployed on*". Examples of hardware platform components are: CPUs (single and multi-core), FPGAs, RC-analog filters, etc. Examples of SW platform components may be considered: RTOSs, Drivers, Ethernet Stacks, etc.

A *PlatformConnector* entity serves to model the connections between platform components. It can be HW or SW, as modelled by the *PlatformConnectorType* entity. Examples of platform connectors are: CAN bus or Ethernet. In order to be bound by a connector, platform components have to be able of playing two different roles: named Target and Source. If a platform component plays both roles, it can be bound by the same connector. This is, for instance, an operational amplifier or flip-flop when connected in feedback.

Component. A *Component* entity is meant to abstract the model of an application component. In order to support MULTIPAR, a component consists of an Interface and one or more Variant entities.

The *Interface* entity models the functional properties of the component, i.e. the component interface is divided into the required and provided interfaces. The component interface is defined through a number of ports. In Figure 1, a port is represented by the *Port* entity. It is modelled by assuming that it can play two roles, named *InPort* and *OutPort*, which serves to respectively model the required and the provided interfaces. It is also assumed that each component at least has an input port. A port is also characterized by the mode in which information is exchanged, for instance synchronously or asynchronously.

The *Variant* entity models a deployed component on a platform. Each component may have more variants associated with it. Each variant is characterized by an implementation technology (i.e. SW or HW), a format (i.e. C-code, VHDL-code, binary, etc.), and a platform of deployment (i.e. the operating system, the processor technology, etc.). A variant is specifically defined by the relationship with the data type associated to each port. Although the interface is the same, regardless of the component variants, each variant is characterized by a specific platform-dependent implementation, resulting into a specific port implementation. This is shown in Figure 1 by the *PortTypeBinding* and *PortDataType* relationships. Examples of Port DataType are: Single Analog (e.g. ADC) Single Digital (e.g. I2C, RS232, etc.), Multiple Digital (e.g. 8/16/32-bit BUS, etc.), etc. Further, each variant has associated with it a number of *EFP* entities, meant to model the component's EFPs. An EFP entity is characterized by: (i) a category e.g. component execution time, component size, component reliability, etc.; (ii) a subcategory, which might be needed to distinguish EFPs specific for HW or SW components (e.g. for the component size category, examples are: the memory footprint for a SW component and the number of gates for a HW component); (iii) the context under which the property value is set; (iv) a collection of values for the given property.

Connector. A *Connector* models the binding between the components. The bindings are realized through the interfaces. Thus, the connector always binds two ports. These must be an input and an output port respectively, even if belonging to the same component (e.g. for implementing feedback signals). If the same component is connected through an input port and output port, it must be able to play both roles, i.e. the Source and the Target - Figure 1. An output port can be associated to more than one connectors.

III. THE MULTIPAR PROCESS

The MULTIPAR process includes a set of activities which lead to the HW/SW partitioning of the application components. The process adopts a Component-based Development (CBD) process with emphasis on components' reuse. The MULTIPAR process is performed in several iterations of two phases: the first one building the system architecture with focus on the functional requirements and the second one considering extra-functional requirements.

During the process several artifacts are used as input and some new artifacts are being created. A *list of application*

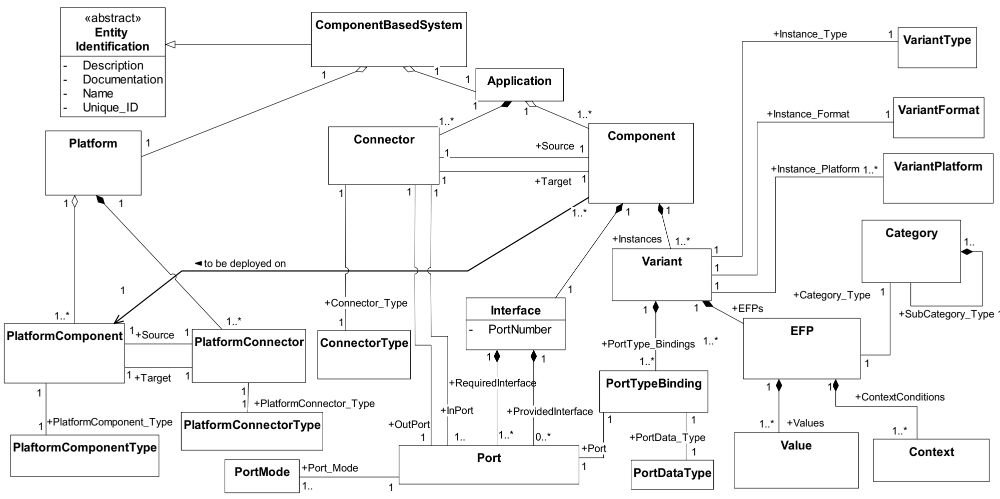
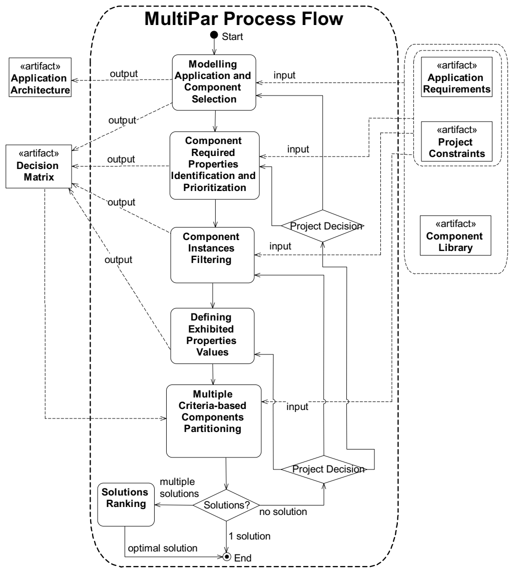


Figure 1. Main view of the CBS Metamodel for enabling software and hardware partitioning

requirements and a *list of project constraints* are used as input to the analysis, design and architecting activities. In addition, information about existing components is used, from a *component library*. The components in the library are compliant to the metamodel defined in Figure 1.

The MULTIPAR process flow described by Figure 2, embodies the following activities:



MultiPar Process Flow
Start
«artifact» Application Architecture
«artifact» Decision Matrix
Modelling Application and Component Selection
Component Required Properties Identification and Prioritization
Component Instances Filtering
Defining Exhibited Properties Values
Multiple Criteria-based Components Partitioning
Solutions?
Solutions Ranking
multiple solutions
no solution
1 solution
optimal solution
End
Project Decision
Project Decision
output
output
output
output
output
input
input
input
input
«artifact» Application Requirements
«artifact» Project Constraints
«artifact» Component Library

Figure 2. MultiPar Process Flow Diagram

Application Modelling and Component Selection. The modelling activity is based on the information provided from the application requirements and project constraints. By using this information as inputs, the application is modelled as a set of platform-independent components and connectors binding the components, compliantly with the metamodel defined in Figure 1. If found in the component library, the defined components will be inserted into a matrix called Decision Matrix (DM). In this process all available implemented component variants C_i will be included. They will have different implementations and different properties, but the same interface. The architectural design and components' identification is a complex process and may require several iterations. For each identified component which do not exist, a new component entry will be inserted into the DM. This component entry will include the interface specification and two virtual (i.e. yet non-existing) component variants (a SW one and HW one).

Component Required Properties Identification and Prioritization. After the architecture design and component identification, the component properties that fulfill the extra-functional requirements must be identified. From the extra-functional requirements and project constraints and by the architectural analysis two sets of properties of interest for the partitioning decisions - application properties P_{iA} and project-related properties P_{iP} will be identified and their related values will be defined. These are the required properties of the identified components. The required properties should be prioritized based on a trade-off analysis. The priorities will be used as weight factors for the MCDA. The outcomes of this activity will be saved in the DM. The component variants C_i have some properties, i.e. they exhibit some properties. We define them as exhibited properties P_{iE}. The sets P_{iA} and P_{iP} may include some properties that are not defined in the component variants C_i. The missing values will be setup in a late stage.

Component Variants Filtering. Many of the component variants do not satisfy the specifications of the required properties. For example, there can be a required property that a particular component should be deployed on a particular component platform or an EFP, for example memory usage, is far above the memory allowed to be used, as specified by the corresponding required property. Such variants are not relevant for the application and can be discarded as a candidate, i.e. they can be removed from the DM. The filtering activity is performed whenever the values of the exhibited properties are changed.

Defining Exhibited Properties Values. For the components variants which the P_{iE} do not have the specified values, the values have to be assigned. They might be measured, simulated or estimated. They will populate the DM (and possibly the component library for future reuse).

Multiple Criteria-based Component Partitioning. When the exhibited properties and the required properties are defined as well as the cost function expressed by the required properties priorities it is possible to search for a (local) optimal partitioning solution. Different MCDA techniques can be used in the solution provision. This activity might converge to a single or several solutions or no solution can be found. In a case it converges to a single solution the process flow is concluded. If several solutions are available, they will be ranked. If it will not converge new iterations need to be carried out.

Solutions Ranking. This activity is the last one and is supposed to be performed in case several partitioning solutions are available. Based on project constraints and application requirements, further decision criteria will be defined for enabling MCDA-based ranking.

Figure 2 does not include the analysis and verification activities; they are iteratively performed along to each of the above described activities.

IV. Wind Turbine: Industrial Case Study

To show the conformability of a model from the proposed metamodel and the feasibility of MultiPar in an real industrial application we have developed a simple wind turbine application (WTA). The main objective of the WTA is to control the transformation of the rotational mechanical energy of the rotor blades, caused by the wind, into electrical energy, which will be redistributed via a power network. The core element of the application is the controller, which dynamically regulates the rotor blades at different wind profiles while maximizing the generation of electrical energy and avoiding any damage to the plant. The application is deployed on an industrial wind turbine prototype.

In this case study we illustrate two simple different deployment scenarios of the same WTA. Both scenarios are driven by different sets of required properties. They are named respectively: **Performance-driven scenario** and **Effort-driven scenario**. The first scenario is originated by the need of satisfying application requirements on components execution time. The second scenario needs to satisfy properties derived by project constraints such as the the effort for the technology-dependent design of components, adaptation of existing components, the testing, etc.

The WTA is characterized by a set of platform project constraints, one of them specifying a combined technology implementation: CPU and FPGA. A solution to this comes in the form of the Xilinx ZynQ chip.

The Wind Turbine MultiPar Process Flow. We briefly describe in the following the process flow behind the realization of the use case.

1. *WTA Modelling and Component Selection*. Based on the WTA requirements and project constraints lists, as well as on the information available from the existing components, the application architecture was defined. It is modelled as a number of interconnected components. The model is conformed to the proposed metamodel, and it is implemented by using The MathWorks Simulink. The simplified model is shown by Figure 3. The application is decomposed as follows: the *SensorInterface* (C1), interfacing the feedback signal coming from the sensors to the controller, i.e. the turbine speed (TS) and wind speed (WS) signals; the *Filter*, filtering the feedback signals (C2), the *Main Controller* (C3) orchestrating the overall control of the application; the *Pitch Estimator* (C4) estimating the desired pitch angle at the rotor blades; *Pitch Regulator* (C5) regulating of the pitch angle based on the desired pitch and the calculated pitch; *Park and Brake Controller* (C6) setting the pitch command for steering the rotor blade; and *Supervision System* (C7) supervisioning the execution of the overall application. Each component defined by its interface: input and output port, and ports are bound via connectors. For instance, the C3 interface is defined by the *Filtered_TS inport* and the *Pitch_Reference outport*. Example of connector, is given by the *Pitch_Brake_Con*, connecting C3 and C6. Each component has also an entry in the DM, as shown in Figure 4.

In order to validate the design of the application, the model has been simulated, using the plant model (represented in Figure 3 by the grey boxes) which is calibrated against a real wind turbine prototype.

The component library includes a set of variants (i.e. reusable components) for the components C1, C2, C3 and C4 and they are directly populating the DM (see Figure 4), whereas for new components, in this case C5, C6 and C7 two virtual variants: HW_v and SW_v are associated and the related EFPs values are estimated. In addition, for C4 an HW_v (virtual variant) is associated, this is due to a requirement on performance.

2. *Wind Turbine Component Required Properties Identification and Prioritization*. Based on the WTA requirements and project constraints, the P_{iA} and P_{iP} sets were defined. Examples of identified required properties belonging to the P_{iA} sets are: (i) Max Execution Time, (ii) Required Accuracy. Whereas, examples of identified required properties belonging to the P_{iP} set are: Design, Implementation, Testing, Maintenance Effort, etc. Based on design/project

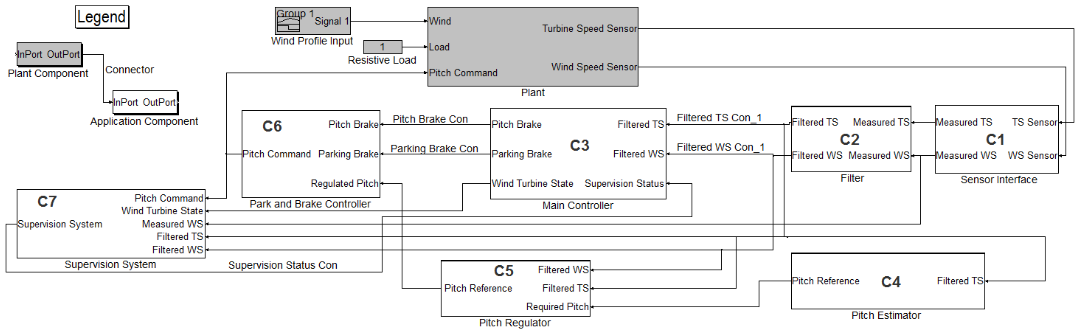
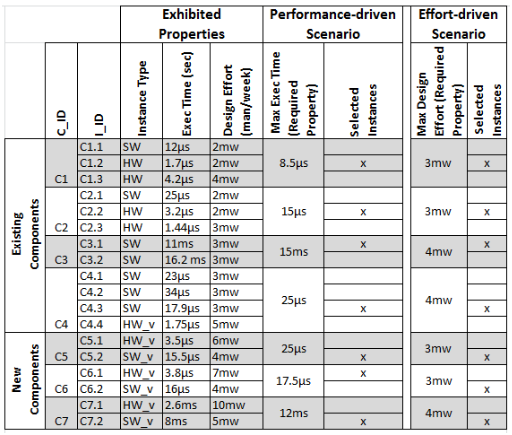


Figure 3. Wind Turbine Application (WTA) Model



			Exhibited Properties			Performance-driven Scenario		Effort-driven Scenario	
	C_ID	I_ID	Instance Type	Exec Time (sec)	Design Effort (man/week)	Max Exec Time (Required Property)	Selected Instances	Max Design Effort (Required Property)	Selected Instances
Existing Components	C1	C1.1	SW	12µs	2mw	8.5µs		3mw	
		C1.2	HW	1.7µs	2mw		x		x
		C1.3	HW	4.2µs	4mw				
	C2	C2.1	SW	25µs	2mw	15µs		3mw	
		C2.2	HW	3.2µs	2mw		x		x
		C2.3	HW	1.44µs	3mw				
	C3	C3.1	SW	11ms	3mw	15ms	x	4mw	x
		C3.2	SW	16.2 ms	3mw				
	C4	C4.1	SW	23µs	3mw	25µs		4mw	
		C4.2	SW	34µs	3mw				
		C4.3	SW	17.9µs	3mw		x		x
		C4.4	HW_v	1.75µs	5mw				
New Components	C5	C5.1	HW_v	3.5µs	6mw	25µs		3mw	
		C5.2	SW_v	15.5µs	4mw		x		x
	C6	C6.1	HW_v	3.8µs	7mw	17.5µs	x	3mw	
		C6.2	SW_v	16µs	4mw				x
	C7	C7.1	HW_v	2.6ms	10mw	12ms		4mw	
		C7.2	SW_v	8ms	5mw		x		x

Figure 4. Wind Turbine Decision Matrix (DM) - simplified overview

teams expertise, the related value are assigned. With respect to the prioritization of the required properties, different weight values are assigned to the required properties for each scenario.

3. *Filtering Component Variants*. In the next activities the irrelevant variants are not taken into account as follows: C1.1, C2.1, C3.2, C4.2 for the **Performance-driven scenario**, since they do not satisfy the required properties with respect to the max execution time, whereas C1.3, C4.4 for the **Effort-driven scenario,** since they do not satisfy the project constraints with respect to the development effort.
4. *Defining values of exhibited properties*. The exhibited properties which do not have associated a values are estimated/calculated. The estimation of the is supported by the application' simulations on the modelled plant and the estimations provided by The MathWorks Embedded Coder Toolbox used for automatic C code generation (e.g. lines of code, execution time for SW variants), HDL Coder Toolboxes for automatic VHDL code generation and by the Xilinx ISE Simulator for execution time. The estimated values are used for both scenarios. For existing components, some design effort is needed anyway to get them adapted. 5. *Multiple Criteria-based Component Partitioning*. Based on the information available on (i) the DM; (ii) the criteria derived by overall application requirements and project constraints; (iii) design/project team expertise; and by carrying out a visual analysis of the DM the partitioning decisions are taken as follows: for the **Performance-driven scenario.** the Filter, Sensor Interface and Park and Brake Controller components are deployed as hardware, while the remaining components as SW. The need of satisfying the execution time is the main driver of these partitioning decisions. For the **Effort-driven scenario,** the Filter, Sensor Interface components are deployed as HW, while the remaining components as SW. The main driver in this case, is the optimization of overall design cost of the components. Figure 5 shows the selected variants for both scenarios. The key difference is given by the Park and Brake Controller component deployment. It is deployed as an HW component in the Performance-driven scenario and as a SW components in the Effort-driven scenario.

V. Related Work

Over the last decades several application architectural partitioning approaches and procedures mostly oriented towards solutions satisfying performance were proposed, e.g. [4],[2]. Comparisons of the most well-known approaches are cited in [1]. Over the time the increase in complexity required new partitioning approaches, which were focused only on few low-level extra-functional requirements: combination of design costs, energy consumptions, performance as discussed in [3], but still no scalable for handling a large number of requirements. In difference to these approaches, we provide a wider-spectrum method able of considering a larger number of requirements and constraints derived by the application and, here new, by the project. In [5], a general approach for performing quantitative analysis of architectural designs based on a well-defined criteria is proposed. The approach enables to quantitatively rate design architectural alternatives based on performance metrics. In particular, the so called Y-chart approach is presented and further discussed in [6], which identifies the three core main elements influencing the choices in finding feasible solutions. It is mainly focused

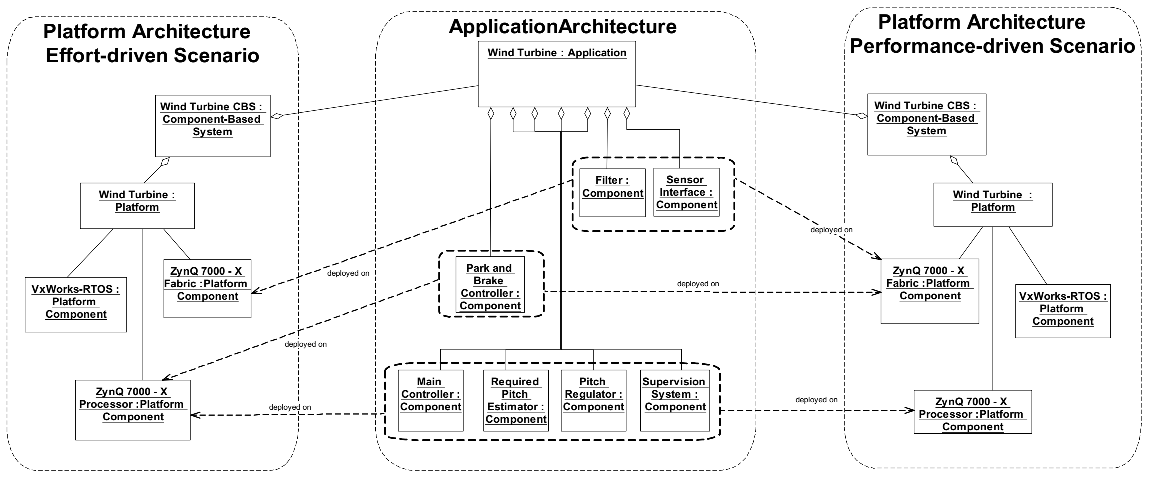

Platform Architecture
Effort-driven Scenario
ApplicationArchitecture
Platform Architecture
Performance-driven Scenario
Wind Turbine : Application
Wind Turbine CBS : Component-Based System
Wind Turbine : Platform
VxWorks-RTOS : Platform Component
ZynQ 7000 - X Fabric :Platform Component
ZynQ 7000 - X Processor :Platform Component
Filter : Component
Sensor Interface : Component
Park and Brake Controller : Component
Main Controller : Component
Required Pitch Estimator : Component
Pitch Regulator : Component
Supervision System : Component
deployed on
Wind Turbine CBS : Component-Based System
Wind Turbine : Platform
ZynQ 7000 - X Fabric :Platform Component
VxWorks-RTOS : Platform Component
ZynQ 7000 - X Processor :Platform Component

Figure 5. Partitioning Deployment Scenarios: Effort-driven (left), Performance-driven (right). Wind Turbine Application Architecture (center).

on performance analysis for a given set of applications, i.e. video-signal processing applications, and further work is left to different domains.

VI. Conclusion

In this paper we have presented a method MultiPar for a systematic and sustainable decision process for partitioning embedded component-based systems into HW and SW. We have presented a metamodel suitable for enabling the partitioning, the method foundation and its process flow.

The novel parts in the method are (i) specification of embedded systems with components as HW and SW implementations, which required some adaptions of the component-based principles, (ii) the model-based process, starting at platform independent level and a offering solution for technology selection enabling high level component reuse, (iii) inclusion of application requirements and project constraints in the partitioning decision process.

For the future work we plan to refine the process, related to the above mentioned challenges, and then to improve the particular activities, in particular choosing the most appropriate MCDA method. Finally our intention is to provide an integrated tool support and evaluate that in a larger industrial context (e.g. in a development of wind turbine control systems in a product line), assuming reuse of the existing components during a larger period.

References

[1] W. Ahmed and D. Myers. Concept-based partitioning for large multidomain multifunctional embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 15(3):22:1–22:41, June 2010.

[2] K. B. Chehida and M. Auguin. Hw/sw partitioning approach for reconfigurable system design. In *Proc. of Int. Conf. on Compilers, architecture, and synthesis for embedded systems*, CASES '02, pages 247–251, New York, USA, 2002. ACM.

[3] R. P. Dick and N. K. Jha. Mocsyn: multiobjective core-based single-chip system synthesis. In *Proc. of Conf. on Design, automation and test in Europe*, DATE '99. ACM, 1999.

[4] R. K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *IEEE Des. Test*, 10(3):29–41, July 1993.

[5] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Proc. of Int. Conf. on Application-Specific Systems, Architectures and Processors*, ASAP '97, pages 338–, Washington, DC, USA, 1997. IEEE.

[6] B. Kienhuis, E. F. Deprettere, P. v. d. Wolf, and K. A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pages 18–37, London, UK, UK, 2002. Springer-Verlag.

[7] M. López-Vallejo and J. C. López. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Trans. Des. Autom. Electron. Syst.*, 8(3):269–297, July 2003.

[8] G. D. Micheli and R. K. Gupta. Hardware/software co-design. *IEEE MICRO*, 85:349–365, 1997.

[9] O. Nierstrasz, G. Arvalo, S. Ducasse, R. Wuyts, A. Black, P. Mller, C. Zeidler, T. Genssler, and R. Born. A component model for field devices. In *Component Deployment*, volume 2370 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2002.

[10] G. Sapienza, I. Crnkovic, and T. Seceleanu. Towards a methodology for hardware and software design separation in embedded systems. In *Proc. of Int. Conf. on Software Eng. Advances*, ICSEA 2012, pages 557–562. IARIA, Nov 2012.

[11] G. Sapienza, I. Crnkovic, and T. Seceleanu. Partitioning decision process for embedded hardware and software deployment. In *Proc. of Int. Workshop on Industrial Experience in Embedded Systems Design, COMPSAC 2013*. IEEE, Jul 2013.

[12] S. Sentilles, P. Stepan, J. Carlson, and I. Crnkovic. Integration of extra-functional properties in component models. In *12th International Symposium on Component Based Software Engineering (CBSE 2009), LNCS 5582*. Springer Berlin, LNCS 5582, June 2009.

[13] J. Teich. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proc. of the IEEE*, 100(Centennial-Issue):1411–1430, 2012.

[14] W. Wolf. A decade of hardware/software codesign. *Computer*, 36(4):38–43, Apr. 2003.