Towards an efficient partitioning decision process for embedded systems

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

ABSTRACT
Development process of advanced embedded industrial automation applications is highly sensitive to an efficient design approach. Widely used approaches in automation industry for design are based on both hardware and software expertise which are not often completely combined in synergic ways. Traditionally design starts too early and gets separated into hardware design and software design at very early stage which negatively impacts the overall application development process due to issues such as mutual flow interruptions and redesigns. In order to minimize or even overcome to the aforementioned issues, this paper presents and discusses a new systematic design approach able of enabling software-independent hardware design as well as hardware-independent software design to a later stage and able of minimizing the hardware and software mutual dependencies after the separation which collectively improves the overall development process.

Categories and Subject Descriptors
K.6.3 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS]: Systems analysis and design, Systems development. C.0 [GENERAL]: Hardware/software interfaces, System architectures, Systems specification methodology.

General Terms
Design.

Keywords
Application Development Process, Partitioning, Decision Criteria, Component-based System.

1. INTRODUCTION
The complexity increasing of embedded industrial applications constantly demands evolvements and improvements of the overall development process. Ideally, the development process has to be able to simultaneously satisfy two main driving requests imposed by the today's market trends: (i) the lowest time-to-market and (ii) the lowest development and product costs, while preserving quality and launching high-competitive products. In addition to the above, the technology advancements in semiconductor and electronics fields coupled with the growing demands of providing more and more sophisticated software functionality constantly challenges the definition of new design methodologies in order to improve the overall development process [1].

Due to the intrinsic nature of embedded systems i.e. the tight coupling between hardware and software, the development process is extremely affected by the efficiency of the design phase which has to rely on methodologies that are able to integrate key paradigms of hardware design and software design in an effective manner. Widely used approaches in automation industry for design are based on hardware as well as software expertise, however not combined in synergic ways. Traditionally the design starts and gets too early separated into hardware design and software design [2] (and these latter have limited mutual visibility) until a late integration in the implementation phase takes place.

In addition to the early design start and separation, due to the heterogeneity of the platforms and functionalities as well as the plurality of requirements and constraints to fulfill, the design of modern embedded industrial application needs to deal with complex architectural problems to solve. As a consequence, the most crucial part of the design is represented by the set of early architectural decisions that need to be taken in order to properly separate the application into hardware and software. Under common practice, separation into hardware or software is an iterative process, approached in a manually controlled “trial and error” mode, which is not supported by suitable and effective tools or systematic decision process. Separation is typically done by invoking individual back-end tools several times in order to later decide which architectural solution appears to be the most suitable one. This approach, unfortunately, is prone to negatively affect the overall application development process due to issues such as flow interruptions and redesigns.

In this paper, we present a new systematic design approach which enables hardware and software design separation as late as possible after a performing a well-structured partitioning decision process. The proposed approach is based on (i) the specification of an application as a component-based system (CBS) [3], (ii) on the identification of a number of component properties such as functional properties, extra-functional properties, project- and application properties that are of key relevance for taking partitioning decision, and (iii) on the identification of techniques
and tools able of satisfying a plurality of decision criteria. By the identification of the set of properties and criteria for decisions, the approach is able of providing a beneficial feedback to the requirements phase. Moreover, it is able to minimize the dependencies between hardware and software after the separation. Collectively the above mentioned methodology features lead towards an overall improvement of the application development process. Specifically, the proposed approach will address embedded applications targeting the automation domain. This is motivated by the fact that automation domain represents still an open field. The concepts presented in this paper are supported by years of experience in industry with design methodologies and embedded systems development. The remainder of the paper is organized as follows. The next section discusses the current state of practice and the research problem. Section 3 describes the research strategy and the main research question. Section 4 describes the new proposed design approach. Section 5 presents the related work. Section 6 concludes the paper and presents the future work.

2. RESEARCH PROBLEM

Typical industrial development process can be often described as a number of sequential phases: requirement (and analysis), design, implementation, verification and validation. The development process starts with the requirement phase in which the requirements are supposed to be identified and analyzed. Immediately after the specification phase the design phase usually branches into two separated design flows: (i) hardware design flow and (ii) software design flow. These latter evolve separately and get into their own implementation. When both hardware implementation and software implementation are completed, the integration takes place. Subsequently the verification and validation phase get in progress. Figure 1 shows a simplified diagram of a traditional development process.

![Figure 1 – Traditional Development Process](image)

In reality, the development process results to be more complex: phases get interleaved and each of them might require be iterated and/or optimized several times over the entire development process. In this scenario the design phase is affected by two main issues which generate side-effects that are negatively impacted to the next phases and the overall development process. In particular, the design phase is subjected to:

- **An early separation into hardware and software flows**: despite the fact that hardware and software for embedded applications are tightly connected, typically the design phase splits too early into the two above mentioned design flows. In this context, hardware and software, after the separation, are considered as two separated entities which are seldom integrated until the integration and verification stages. In principle, (i) hardware does not take into account the computational power required by the software and the capability that the software might offer for enabling hardware optimization and (ii) software does not impact the hardware design specifications, and does not fully exploit the available hardware resources.

As a consequence of the two aforementioned issues generate side-effect such as (i) hardware or software design and implementation interruptions and (ii) hardware and/or software redesigns.

### 2.1 Underestimation of the partitioning

In practice, the too early design start corresponding to the too early flow (hardware and software) separation do not allow to properly focus on the most important and core part of the design phase – referred in this paper as partitioning decision process – process dealing with the decision upon which parts of the application are supposed to be designed in hardware and which are supposed to be designed in software. A relevant work discussing more in details the critical relevance of the partitioning and its fundamental objectives can be read in [4]. In practice, the importance of the decision process is often underestimated and the too early design separation does not allow allocating enough time for it. In addition to the above, the trend of approaching the design using model-driven/based tools (e.g. The MathWorks Simulink®, IBM® Rational® Rhapsody®) well-supports approaches such as “trial and error”, but in practice it hides: (i) the need of focusing on the partitioning decision process before splitting the design into hardware and software and (ii) the importance of having appropriate process that are able of guiding and supporting the partitioning decisions.

3. RESEARCH STRATEGY AND QUESTION

In order to present the overall research strategy and research questions we follow a methodology which has in foundation in the guidelines proposed by Shaw in [5]. By Figure 2, we intend to describe the strategy flow of our research:

- **Practical Problem and Solution**: in Section 2, we have identified and discussed what are the practical problems associated to a typical development process during the design phase. We identified that the application partitioning process into hardware and software is a complex problem which today is performed inefficiently and inaccurately. As highlighted this is the originated by the early start of the design and the early separation into hardware and software flows. This leads to issues such as flow interruptions and redesign which negatively affect the overall development process in term of quality, efficiency and costs. We aim to provide solutions able of pushing the application partitioning to a later stage.

![Figure 2 – Strategy Flow Diagram](image)
4. PARTITIONING DECISION PROCESS PROPOSAL

In this Section, we present a new partitioning decision process capable of reducing or even overcoming the issues previously presented and discussed. We proposed a systematic approach which is mainly characterized by two following key features:

- providing support to the requirement phase
- enabling *software-independent* hardware design and *hardware-independent* software design separation as late as possible

### Figure 2 – Research Strategy Overview

- **Idealized Problem and Solution:** due to the fact that the practical problem is quite complex, the real problem is transferred into a research setting context. In this context, the problem will be investigated and formulated, and the aim is to find solutions to it. Formulated as a question the main research goal can be stated as follows: *How to improve the partitioning decision process by executing it as late as possible and based on a well-defined set of criteria?* Our main research objective is to propose solutions towards establishing an efficient partitioning decision process at late stage by providing process, methods and tools.

- **Research Products:** the main outcomes of the research will be the definition of the overall process, methods and tool suitable for enabling a well-defined late partitioning.

- **Research Validation:** in order to validate the proposed solution, our strategy will be based on questions related to the feasibility and the efficiency of the proposed partitioning decision process:
  - *Is it possible to achieve our main research goal? (Feasibility-question)*
  - *Is the new proposed process really more efficient than the traditional one? (Efficiency-question)*

Related to the feasibility-question, the proposed process is intended to be validated by the development of a prototype as well as a real application. It will be performed in two steps:

- **1 step validation:** development of a prototype, which will be used to validate the solution to the Idealized Problem
- **2 step validation:** development of real product which is supposed to validate the solution to Practical Problem (this part of the validation will be out of scope for this research)

Related to the efficiency-question, the proposed approach will be evaluated in different ways:

- By performing experiments with students
- By using the some professional experts in order to collect their feedback
- By starting a pilot project, using it as a case study, and following it up

### Figure 3 – New Approach Proposal

Figure 3 shows a diagram describing the new approach proposal. In our approach the application is considered as a CBS system [3]. In order to guide a well-structure partitioning decision process, our approach will:

- Build a component library able of providing support for the design the application as such systems.
- Extend the components properties from a project as well as application perspective
- Define a number of decision criteria from a project as well as application perspective
- Identify techniques and/or tools able of supporting multiple criteria decisions.

#### 4.1 Application and Component Formalization

In our research, an application is treated as a number of hardware and software components, which are interconnected. In order to formally provide a definition of (i) application, (ii) hardware and software components, and (iii) interconnections, we intend to adapt the definition proposed in [3] for software components. It will cover hardware components as well. Hence, the application can be seen as a component-based system (CBS). By adapting the definition provided in [3], this can be described by:

$$CBS = (C, B, P)$$

where

- a set of components which can be hardware or software ($C$);
- a set of bindings interconnecting the components ($B$);
a platform on which deploying the set of components (P);

By extending the definition provided in [3] a hardware or software component (C) can be described by:

- an interface (I)
- a set of properties (P)

\[ C = \langle I, P \rangle \]

The interface defines a number of components functional properties, and allows the components to interact with other components. From a functional point of view, we distinguish between a required interface and a provided interface. The required interfaces (RI) can be seen as a collection of input functions to the component. The provided interface (PI) can be seen as a collection of output functions from the component.

\[ I = \langle RI, PI \rangle \]

The set of properties (P) serve to specify a component from a non-functional point of view, defined in [3] as extra-functional properties (EFP). These properties can be originated by different source, e.g.:

- Properties exhibited by the implementation (e.g. execution time, memory size, etc.)
- Properties related to modeling such as for instance: reliability, upgradability, maintainability, etc.
- Properties exhibited by the context (e.g. platform, technology, tools, etc.)

Time wise, new properties might need to be taken into account. As a consequence, the number of elements belonging to P can increase over the time. This is formally defined as follows:

\[ P = P_0 + P_{new} \]

where \( P_0 \) represents the initial component properties before a new property, defined as \( P_{new} \), is added to the P set.

From an architectural point of view, the bindings serve to describe the connection of the component with other components via the required and provided interface.

The set of bindings can be divided into two classes

- the input binding (IB) to the components which is occurs via the required interface;
- the output binding (OB) to the components which is occurs via the provided interface;

\[ B = \langle IB, OB \rangle \]

The application so formalized can be model by using a model. The latter is meant for delineating a formal specification of the application in term of architecture, functions and behavior within the given domain. In this case, the above defined CBS can be modeled by using number of interconnected component models (CM). A component model is a merely representation of a real hardware or software component, formally defined in [3]. Figure 1 shows an example of application architectural model which consists of interconnected CMs. The set of bindings between components is represented in the Figure 4 by using connectors.

**Figure 4 – Application Architectural Model, Components Model and Components Instances**

An implementation of a CM is here defined as component instance (CI). Each CM may have associated one or more instances of a previous implementation (as shows in Figure 4 for CM8) or it may do not have any already existing implementation associated (as shown in Figure 4 for CM5). The connectors between CMs serve to model the component bindings.

**4.2 The Library of Components**

The first step for achieving our main goal is to build a library of CMs to support the requirement phase, the application design and the partitioning decision process. We named this library as Component Library (CL). The CL will contain all of the CMs and related CIs that has been part of previous application development process. In order to build the library, we need to perform the following steps:

A. Analysis of requirements and designs of previous applications related to the automation domain in order to formally identify hardware and software components. Hardware and software components will be identified according to the specification of their architecture, functions and behavior within the application context. In addition to the above, their interface, properties will be analyzed and identified as well. Each property will be associated to a label, in order to facilitate the searching operations in the library.

B. Specification of the library in term of its design and the definition of the library operational rules. More in details, each entry of the CL will have associated the following information: (i) the CM identifier (CM_ID), (ii) the CM interface (I) as defined in Section 4.1, (iii) the component instance identifier set (CI_ID) (a CM can have more than one CI associated), (iv) the CM properties as defined in Section 4.1. Formally, a CM entry in the CL can be described as follow.

\[ CM_{entry} = \langle CM\_ID, CI\_ID, P \rangle \]

In addition to the above specification, it will be necessary to define a number of operational rules for working on the library (e.g., for inserting, and searching operations) will be provided. Time wise, the library is supposed to be updated every time a new CM or CI will be available. This may imply that new properties have to be taken into account and they have to be evaluated.
with respect to all of the already existing CMs. An example showing a possible implementation of the CL is presented in Figure 5.

<table>
<thead>
<tr>
<th>CM_ID</th>
<th>NAME</th>
<th>REQUIRED INTERFACE</th>
<th>PROVIDED INTERFACE</th>
<th>CI</th>
<th>CI ID</th>
<th>RELIABILITY</th>
<th>RELIABILITY VALUE</th>
<th>PROPERTIES (EFP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>PI Controller</td>
<td>Set Point Signal</td>
<td>Control Signal</td>
<td>C1</td>
<td>High</td>
<td><em>W</em></td>
<td>9 µs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM2</td>
<td>Current Filter</td>
<td>Measured Current</td>
<td>Filtered Current</td>
<td>C2</td>
<td>Medium</td>
<td><em>W</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5 – Example of Component Library (CL)**

We intend to have a 2-dimensions table for an easy management, analysis and extraction of the CMs.

C. Gathering of the existing CMs into the library, including the information related to its component instances (CIs). As shown by Figure 5, for instance CM1 has several CIs associated, this is the case of CM1 which has four CIs associated.

### 4.3 Application Modeling and Decomposition

At model level, based on several aspects e.g. functional and non-functional requirements, human factors (e.g. designers’ expertise and knowledge, etc.), constraints (e.g. legacy, reuse of existing platform, tool chains, manufacturing platforms and cost, etc.) technology, etc., designers have to decompose an application into a number of CMs. Before reaching a mature stage from where to start the partitioning decision process, the modeling and decomposition of the application might require several steps and iterations. The granularity of the decomposition is dependent from the designers’ expertise and their judgment in relation to the overall project constraints. In our approach, the application modeling and decomposition should get support by the analysis of the already available CMs in the library. Designers can use the library for different purposes, for instance to get information about the interface of a specific CM or to get property information of a CM before modeling it. During the modeling and decomposition each CM is supposed to be defined in terms of its interface (I) and properties (P). When the designers consider the application model ready to go through the decision partitioning process, all of the applications CMs need to be further analyzed with the regards to the CL, in order to extract from the library the already existing CMs that have to be processed. This implies that all of the application CMs will be divided into two sets as follows:

- **Set A**: it will contain all of the application CMs which can be mapped into a corresponding CM already available in the library.
- **Set B**: it will contain all of the application CMs which do not have any existing CM available in the library.

Following the same approach proposed for building the library, the application CMs can be expressed in form of table as well, as shown by Figure 6 in Section A.

It provides an example of the application CMs sorted into the two sets (Set A and Set B). The CMs belonging to Set A are extracted by the library. The CMs belonging to Set B do not have any entry in the library. Moreover, it can be highlighted that the CMs belonging to Set B might do not have any implementation available yet. This is not valid in case of legacy components that are not included of the library yet and which can have already an implementation. In addition to this, it can be noticed that compared to Figure 5, two CMs belonging to Set B (i.e., CM56 and CM91) extend the set of the component library properties with a new property labeled ‘Accuracy’.

### 4.4 Horizontal Filtering and Weighting Operations

Before proceeding towards the partitioning decision process, it might be needed to perform a number of filtering and/or weighting operations on the application CMs. These operations are relevant in order to eliminate CIs (i.e., filtering operation) that are not relevant to be taken into account or to properly emphasizing the contribution (i.e., weighting operations) of one or more EFP properties for a particular CM that has an impact on the final partitioning results.

#### 4.4.1 Horizontal Filtering Operations

By horizontal filtering, we meant to classify all of the filtering operations that act on a single component. These types of filtering operations affect the row of the table provided in Figure 6. An example of a filtering operation might be given by the removal of a specific CI from a CM belonging to Set A, considered to be not relevant for the development of this application.

#### 4.4.2 Horizontal Weighting Operations

By horizontal weighing, we meant to classify all of the weighting operations that act on a single component. An example of a weighting operation may be given by the assignment of a high weight to a component belonging to Set B which has to be implemented in HW due to legacy reasons.

Horizontal filtering as well as weighting operations has to be identified and performed either manually or automatically by the designers. These operations are strictly depended by the project and application, so they can be different from other application development process.

### 4.5 Properties and Criteria related to Project and Application

When the designers decide that the application decomposition into CMs has reached a satisfactory level of granularity, all of the applications CMs need to be further evaluated from a project as well as application perspective. It is relevant for the partitioning decision process to identify CM properties and overall criteria derived (i) by the project estimations and constraints as well as (ii) by the application requirement and constraints.

#### 4.5.1 Project and Application Properties

Project and application properties are supposed to further characterize a single CM. From a project point of view, designers have to identify what are the most relevant properties associated
to a CM that affect the partitioning decisions and they have to define the range of variation of these properties (e.g., low, medium, and high) as well. For instance, typical project properties that can be identified are: (i) time-to-design a CM, (ii) time-to-implement a CM, (iii) implementation priority, etc. Indeed, from the application perspective, designers have to identify what are the key properties associated to a CM in relation to the overall application requirements and constraints. Also, in this case, it is important to establish the property variation ranges. Example of application properties we may think of are: (i) the max execution time, (ii) the level of upgradability of the components or (iii) the level of required reliability for the components. In Section B and Section C of the table provided in Figure 6, examples of project properties and application properties are shown. The complete table, which includes all of the three Sections (A, B and C) and the Set A and B, is here named ‘Partitioning Decision Process Table’. Formally, by extending the definition of CM entry provided in Section 4.2, we can define each entry of this table, here named CM_to_be_processed, as follows:

\[
CM_{to\_be\_processed} = \langle CM\_ID, I, CI\_ID, P, PP, AP \rangle
\]

where PP represents the identified set of project properties and AP represents the identified set of application properties.

### 4.5.2 Project and Application Criteria

As part of the partitioning decision process it is important to identify the project and application criteria that condition the process itself. These criteria can be (i) either related to the project or application properties of a CM or (ii) either related to the all CMs in total. In the first case, as soon as all of the project and application properties are identified, it is important to define the criteria for assigning a value related to a property to each CM. An example of criteria we may think of follows: the sum of the time-to-design for each CM does not have to exceed a fixed number of hours (h), for instance 1000h. In case the criteria cannot to be satisfied, the designers have to re-evaluate and eventually redefine the identified properties as well as the already stated criteria. For instance, it may happen that by assigning to each CM the time to be designed, the sum of the overall time-to-design will largely exceed the fixed number of total hours, e.g., it will be equal to 1500h. In the second case, the criteria has to be identified in relation to the overall CMs and might be not related to any project or application properties. Example of such type of criteria that we may think of are:

- it may be required that 1/3 of the application CMs have to be implemented in software.
- it may be required that a particular technology has to be used for product retrofitting reasons

The objectiveness and effectiveness in identifying the properties and the criteria related to the project as well as the application, is of crucial importance for assuring the quality and reliability of the partitioning outcomes.

### 4.6 Vertical Filtering and Weighting Operations

Assuming that all of the project as well as application properties have been identified, and assuming that the related value has been assigned to each application CM, the next step is to perform a number of filtering and/or weighting operations before processing the CMs_to_be_processed. As already highlighted in Section 4.4, these operations are relevant in order to eliminate (filtering) CIs or properly emphasizing (weighting) the contribution of EFP, PP and AP on the final partitioning results.

#### 4.6.1 Vertical Filtering Operations

By vertical filtering, we meant to classify all of the filtering operations that act on a group of components based on one or more properties. These types of filtering operations are acting on the CMs by the columns of the table in Figure 6. An example of a filtering operation may be given by the operation of remove of all of CIs implemented on a specific platform from the CMs_to_be_processed, since considered to be not relevant for the development of this application.
4.6.2 Vertical Weighting Operations
By vertical weighing, we meant to classify all of the weighting operations that act on a group of components based on one or more properties. An example of a weighting operation may be given by the assignment of a higher weight to a subset of project properties.
Vertical filtering as well as weighting operations has to be identified and performed either manually or automatically by the designers.

4.7 Techniques and Tool Chain Support
In order to support the overall partitioning decision process, we need to identify if there are available techniques and tools able of effectively making decisions among numerous, complex and even conflicting criteria. To achieve it, we intend to perform the following steps:

A. Identification of the requirements those techniques and/or a tool (or collection of tools) has to fulfill in order to support the partitioning decision process. Example of requirements we may think of are:
- able of handling large variety of CMs of different nature
- able of providing mechanisms to assign a criticality-level to a CM (or group of CMs)
- able of supporting the decisions uncertainties and/or missing values
- able of providing measurement and for prioritization
- able of tracing-backward the decisions taken
- able of ranking solutions in case of multiple possibilities

B. Identification and analysis of existing techniques and/or tools that fully or partially meet the identified requirements.

C. Selection of techniques and/or tools in case several will be available.

D. Definition and establishment of how the selected techniques and/or tools have to operate in order to allow the partitioning decision process.

4.8 Partitioning Decision Process Flow
Assuming that the library has been already built, which implies that several applications development process in the automation industry domain have been analyzed, relevant EPF identified and as a consequence a number of CMs and their associated CIs are available and ready to get accessed via the library. In addition, assuming that techniques and tools have been identified, selected and their operations in relation the partitioning decision process have been established, we describe the proposed overall partitioning decision process flow as follows:

A. Modeling and decomposition of the application into a number of component CMs. Each CM has to be fully described in term of interface and properties. The modeling and decomposition can get support by the information available in the library, as described in Section 4.3. By using the library, we provide to designers a mean (i) to take into account previous expertise; (ii) to provide effective feedback to the requirements engineers in case of incompleteness or lack of requirements; (iii) to support and speedup the modeling by CM reuse.

B. Classifications of the application CMs into two sets (referred as Set A and B in Section 4.3) in order to distinguish between which CMs have entries in the library and which CMs do not have any entry in the library. Analysis of the CM properties of the CMs belonging to Set B, in order to identify new properties that are not part of the library. If necessary, all of the CMs belonging to Set A have to be re-evaluated in accordance with the new identified properties.

C. Performing of the horizontal filtering and weighting operations as described in Section 4.4. These operations can be performed manually or automatically if a tool or more tools have been already identified and selected for accomplishing them.

D. Identification of the project and application properties, including the definition of each property variation as presented in Section 4.5.1. This part of the partitioning decision process is of key relevance and it varies from project to project and from application to application.

E. Identification of project and application criteria as presented in Section 4.5.2. This part of the partitioning decision process is of key relevance and it varies from project to project and from application to application.

F. Performing of the vertical filtering and weighting operations as described in Section 4.6. These operations can be performed manually or automatically if a tool or more tools have been already identified and selected for accomplishing them.

G. Applying the identified techniques and/or tools (i) to perform the decisions upon the application is partitioned into hardware and software CMs which later will be deployed on the platform. The expected outcome is a partitioning solution or even several solutions which have to be able at the same time of (ii) satisfying the properties and criteria defined at point D and (iii) fulfilling the criteria defined at points E. The solution can be seen as a contribution of two parts: (i) respect to the CMs belonging to Set A and (ii) respect to the CMs belonging to Set B. In the first case the expected solution is a selection of the most suitable CI for each CM. In the second case the expected solution is a sorting of the all of the CMs into hardware or software.

H. Ranking of the solutions or process iteration if needed. In case of multiple feasible solutions, they are supposed to be ranking according to further criteria defined by the designers. In case there are no feasible solutions the process need to be iterated starting from point D.

5. RELATED WORK
Partitioning of application into hardware and software is generally considered to be a NP (non-deterministic polynomial)-Hard problem [6]. It is an extensively studied problem; classical approaches based on heuristic, iterative and clustering algorithms


are presented and deeply discussed in [7]. A sophisticated integer linear programming model for joint partitioning and scheduling is presented in [8]. Additional approaches like Genetic Algorithm and Artificial Neural Network are proposed in [9], [10]. It can highlight that in literature the partitioning problem is treating quite often as nondeterministic polynomial problem to be optimized. Mainly partitioning approach are focused on performance optimization (e.g., of execution time, of power or memory consumption). For instance the proposed approach in [8] presents a partition scheme for achieving partitioning results with low power consumption and fast execution time. Partitioning decisions, today, require to be taken in relation to 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 (i) [12] which use a Multi Criteria Decision Analysis for ranking partitioning solution based on trade-off analysis and by (iii) [13] which presents a partitioning process able of supporting application constraints imposed by reuse of existing modules in the automotive industry. In the paper we have proposed a systematic approach which is able of performing partitioning over multiple project and application criteria before the separation into hardware and software design flows. In addition, the proposed method is able of supporting re-use of existing components already. The combination of both features makes the proposed approach new.

6. CONCLUSION AND FUTURE WORK

The development process tailoring industrial automation embedded application is negatively affected by the early partitioning of the application into hardware and software. In this paper we have discussed the current state of practice and the related issues. Our main research objective aims to find solutions able of pushing the application partitioning to a later stage and enabling hardware and software independent design after separation. In order to reach our objective we proposed a new well-defined partitioning decision process based on (i) the formalization of the application as a CBS, (ii) the identification of component properties and selection criteria from the project as well as the application point of view, (iii) techniques and tool chain able of supporting multiple criteria decision process. The next step for our research work is to build the component library. In order to perform it we intend to identify and analysis previous related issues. Our plan to validate the proposed approach is to build the component library, as well as of component properties and selection criteria, before the separation into hardware and software design flows. In parallel we need to identify the techniques and/or tools that might be able of supporting the partitioning decision process. Immediately after we plan to validate the proposed approach on the industrial prototype available within the framework of the Artemisia iFEST project [14].

7. ACKNOWLEDGMENTS

This research is supported by the Knowledge Foundation (KKS) through ITS-EASY, an industrial research school in Embedded Software and Systems, affiliated with the School of Innovation, Design and Engineering (IDT) at Mälardalen University (MDH), Sweden.

8. REFERENCES


