Abstract—The development of Critical Embedded Systems (CES) like Unmanned Aerial Vehicles (UAV) is complex because it needs to ensure a high degree of quality, with affordable cost and delivery time. It is also necessary to ensure security since failures in this type of system can lead to catastrophic results. In this sense, a Model-Driven Development (MDD) approach presents itself as a good alternative to the traditional development because coding complexity will be reduced by the use of high level models. In addition, it avoids the introduction of coding errors by human programmers, since the critical code will be built automatically through models transformation. From another perspective, Embedded Systems Development can benefit from Software Engineering techniques like Product Lines to reduce costs and time-to-market. While other works propose the use of Product Line techniques to improve Embedded Software development, we propose a Product Line approach to the whole Critical Embedded System development life cycle, including hardware variability management. Therefore, this paper proposes a Critical Embedded System Product Line Model Based approach, which aims to reduce the above mentioned challenges. The development approach proposes a Domain Engineering and Application Engineering focused on the system, with both software and hardware. To illustrate the proposed approach we include some artifacts from a case study in the UAV domain.

I. INTRODUCTION

Embedded systems development approaches are constantly evolving to meet the growing need for new and more complex applications. In the embedded systems domain we can highlight CES, which are embedded systems whose failure could result in loss of lives or on significant environmental or property damage. CES are common in medical devices applications, aircraft flight control systems, weapons, and nuclear systems [1]. This kind of systems breaks with traditional systems engineering, as they bring together technical requirements such as: criticality, need for low power or fuel (autonomy), reactivity, robustness, scalability with guarantee of quality and functionality at affordable costs [2].

Since hardware evolves fast, the new possibilities created by their evolution increase users expectations for new functionalities to be realized and integrated into an embedded system. However, software development does not cope with the hardware evolution and consequently system development either. To build CES to meet the mentioned requirements in an acceptable time-to-market, we need new efficient and flexible development methodologies that can reduce design complexity.

Among several possibilities, Software Product Line Engineering seems a promising one, as it brings great benefits such as: reducing system costs, because the systems are designed collectively and generated automatically or semi-automatically through the composition of reusable artifacts; time-to-market reduction, since this technique allows to design systems in such a way to predict the possible variabilities; and increasing system reliability, because the systems are assembled from reusable and extensively tested resources and many other advantages [3]. However, for the effective use of this technique in the CES domain we should adapt it for System Product Line Engineering (SPLE), i.e., to apply product line techniques to design the system and not only the software. In conjunction with SPLE we propose the use of model-driven development (MDD) techniques to increase the level of abstraction by means of models and to allow automatic generation of system product line artifacts through models transformations.

Therefore, this paper aims to define and present a complete approach for the design of CES using a combination of both System Product Line and MDD techniques. The differential of the proposed approach are: the use of MDD in both Domain and Application Engineering, the management of both software and hardware variabilities, and the control of software and hardware variability dependencies. For an effective use of MDD we propose the use of the MARTE profile [4] in conjunction with SysML [5] and a simple variability management stereotypes proposed here. Finally, to illustrate the various concepts presented in the proposed approach we include some artifacts from a case study in the UAV domain.

The rest of this paper is organized as follows: Section II presents a background summary of Product Line Engineering and Model-driven Development; Section III illustrates some related works; Section IV presents the proposed approach, followed by a description of the examples by means of a UAV product line case study; lastly, Section V presents the conclusions of this paper.
II. BACKGROUND

According to Northrop et al. [3], a Software Product Line is a set of software systems that share common and managed features satisfying the specific needs of a particular market segment or mission and that are systematically developed from a common set of core assets. The products of a PL distinguish from each other in terms of features, which are user-visible aspects or characteristics of a software system or systems [6].

On the other hand, in Model-Driven approaches, the software complexity concentrate on high level models and not in the code, which can be automatically generated from the models. Furthermore, one can improve the system quality with the use of V&V (Verification and Validation) methods [7]. According to model-based approaches, models become part of the final product and most of the development complexity shall belong to the transformations that should be used to automatically or semi-automatically produce code (or pieces of code) by transformation tools.

These techniques offer an interesting solution to the challenges in the CES domain because they have great synergy, since they propose automation, raising the level of abstraction and reuse. Additionally, they are being widely used in various academic research and industrial projects as discussed in the next Section.

III. RELATED WORKS

While there is a large number of researches that make use of either PL and embedded systems or MDD and embedded systems, due to space limitations, it is not possible here to give an exhaustive description, thus we only provide a brief summary of some works that make use of PL and MDD for the CES domain, similarly to the approach proposed in this work.

Eriksson et al. [8] present a work that focuses on the system (hardware and software) and introduce a feature-based and use case approach for the process of system requirements. While the focus of these authors is only the step of requirements, the work presented in this paper focuses on the entire System Product Line development life cycle.

In the thesis described in Habli [9], the author defines and evaluates a model-based approach to ensure systems and process in a CES Product Line. The approach is based on a safety metamodel that captures the necessary dependencies between the safety plan, the safety assessment and the developed artifacts in the PL. The focus of that work is to ensure systems and processes.

The work presented by Polzer et al. [10] is concerned with the variability in control systems software. In their work, they present a model-based PL engineering process using Rapid Control Prototyping system combined with MDD techniques. The authors modularize the components parameterization in a separate setup, which is isolated from the model that defines the behavior of the controller. The authors use Simulink for the control modeling and automatic code generation and Pure::variants [11] to obtain the products, which are proprietary tools.

Regarding the development of critical embedded applications product lines, there are approaches such as ProLiCES [12] and SyMPLES [13]. ProLiCES creates a parallel path in the process to handle the PL domain engineering and also proposes the use of Matlab/Simulink as MDD technique, which limits requirements analysis and concentrates the MDD only in one step of the process. SyMPLES is an approach for PL application in embedded systems through the extension of SysML language to include variability together with a development process, but in this study the authors do not distinguish between the characteristics of hardware and software and focus on the use of SysML for the architecture description. They both show the application of the approach in the development of a UAV PL.

In short, from the observation and analysis of existing approaches, we note that: some of them require a lot of previous knowledge like the use of non standard profiles, which hinders its adoption; other approaches do not define a specific treatment for hardware variability and its impact in software variability; others are based on the use of proprietary software, such as Matlab/Simulink; and, yet other approaches use MDD in limited parts of the development, specially from project to code or just in application engineering phase.

IV. PROPOSED APPROACH

The approach we propose in this paper is different from the above-mentioned related works, as it focuses on: a simplified variability management, which addresses both hardware variability with its underlying software requirements dependences and vice versa; the use of MDD techniques, like automatic generation of hardware descriptions and embedded software from high level models, for rapid design and specification of CES; models based on UML and the use of SysML and MARTE profiles, because it is the most widely used modeling language, it is easier to understand and is considered as a standard by various free existing tools; the use of MDD in both Domain Engineering and Application Engineering phases, with focus on model-to-model transformations in requirements, analysis and design stages and not only model-to-text transformations in later development stages or just during Application Engineering, by enabling the automatic generation of SPL members.

A brief overview of the proposed approach, is illustrated in Fig 1. As well as the classical PL approaches, the proposed approach has two phases, Domain Engineering and Application Engineering. This approach first differs from the others in the abstraction levels proposed for the Domain Engineering: a system abstraction level, a hardware abstraction level and a software abstraction level, which are detailed later on this section.

Observe that hardware variability could impact directly on software requirements and vice versa. For example, consider the following system requirement in a UAV product line: the system should allow the user to choose between broadcast the images to the ground control station at real time or to record a video (in flash memory, for example). In that case, the
UAV hardware must include a camera. Moreover, for each new sensor added, their corresponding software drivers must also be added. Another highlight is the continuous feedback in the artifacts repository, in which we can store any kind of artifact from both hardware or software types. This feedback can come from updates in Domain Engineering or from new different requirements elucidated from new members modeled in Application Engineering. Also noteworthy is the feedback that may exist inside the system, hardware or software abstraction levels, and among these levels. The feedback is represented by both dashed arrows and double-headed arrows.

Before the Domain Engineering takes place, it is performed a development activities planning, as well as an economic feasibility analysis of the SPL, which will indicate whether or not it is worth to be developed. If the SPL is feasible, then we start the Domain Engineering by modeling requirements in the system abstraction level, as detailed below. It is out of the scope of this work to propose domain analysis techniques, as they can be easily found in the literature. So, existing techniques such as those mentioned in the survey by Prieto-Diaz and Arango [15] can be used.

The Domain Engineering is performed in three abstraction levels and starts mainly from the observation and analysis of existing system on the domain of interest: the system level, where we define the system as a whole by means of requirements definition and specification; the hardware level, in which the hardware specification and hardware variability management take place in conjunction with architecture definition, design of the components and simulation; and, the software level, where we perform the software variability specification and management, architecture definition, subsystems design, simulation, testing, and code generation. Since the proposed approach focuses on CES, various activities of validation and verification permeate the activities of each level. It is important to notice that at the hardware and software abstraction levels a more detailed specification takes place by the eventual allocation with schedulability and underlying model transformations (model-to-model and model-to-text transformations), which are used to bridge the gap between these abstract design models and subsequent design phases, such as verification, hardware descriptions of modeled targeted architecture and generation of platform specific embedded software from independent architecturally software specifications.

1) **System abstraction level:** The system abstraction level starts with the SPL requirements definition by means of a domain analysis, in which the user initially specifies the requirements related to the system product line. For this purpose a SysML requirements diagram with the distinction between functional and non-functional requirements including a simple tagged stereotypes mentioned above to define mandatory, optional and alternative requirements is recommended. Refinements of this diagram can be done as long as necessary, specially after hardware and software specification, to produce a validated artifact that can be included in the SPL repository. In the Application Engineering, when concrete products are instantiated, this document will be matched against the specific product requirements. As illustrated in Figure 1, the artifact repository is update during SPL life cycle for both Domain Engineering and Application Engineering. So, if concrete products have new requirements not covered by the SPL, the requirements diagram can be further reviewed to include them.

After the requirements identification, we propose the definition of SPL features based on the requirements model, by means of feature model [16]. This model also defines the SPL mandatory, optional and alternative features, as well as the constraints that must be validated during the composition of the products. The restrictions are mainly represented by the...
dependencies between features.

In Figure 2, we illustrate the system abstraction level UAV feature model. It is important to notice that this is an initial high level feature model and included both hardware and software features that should be refined in both hardware and software abstraction levels. The decision regarding the implementation of some of these features in hardware or software is done at a later point in the project, which can be even possible at the Application Engineering phase, as it depends on the results of simulations in conjunction with specific restrictions of each product, like for example, execution time, cost, and other factors. Some of the features should be implemented and stored in a software/hardware artifact repository. Regarding the repository to store hardware artifacts, we refer in a logical level, to a hardware models repository.

In the next step the system requirements are converted into use case scenarios and into a system domain model. The use case is described using traditional UML Use Case Diagrams plus the stereotypes to represent mandatory, optional and alternative use cases. The system domain model can be modeled using a class diagram, in which the concepts are represented by pseudo-classes, where we also propose the use of the above mentioned stereotypes. These two modeling concepts are strongly related to the functional high level specification described subsequently. While the use case is needed to obtain a SysML block diagram, the domain model is used to communicate with domain experts for a better understanding about the domain, for validating the specification and for a future definition of a domain specific language by means of UML profile.

To finalize the system abstraction level, each use case is converted into a SysML block (or internal block), for example by applying the MADES methodology [2], with the difference that a mandatory use case is converted to a mandatory block, an optional use case is converted to an optional block, and an alternative use case is converted to an alternative block. After including all the developed artifacts in the repository, we can continue the SPL development by going to hardware or software abstract levels or even to both in parallel. Even though some authors consider MARTE and SysML profiles incompatible, by using the MADES methodology we avoided conflicts related to the two profiles. SysML is used for initial requirements and functional description, while MARTE is utilized for the enriched modeling of the global functionality and execution platform/software modeling [2];

![Fig. 2. A portion of the feature model for the UAV Product Line – system abstract level.](image-url)

2) Hardware and software abstraction levels: In the next step of domain engineering we lower the abstraction level, with the possibility of starting with hardware or software abstraction level, or both in parallel. In both hardware and software abstraction levels, the requirements specification focuses on variability management through feature diagram refinement.

It is not necessary to write another requirements model from scratch, but it might be necessary to update the previous one to maintain traceability between artifacts.

It is important to keep in mind that hardware variability management should concern the impact evaluation of hardware variabilities on software requirements, and also software variability management should concern the impact evaluation of software variabilities in hardware requirements. To manage this impact we propose the use of a dependence matrix, which relates every possible impacting hardware variabilities to the corresponding software change variabilities and, on the other hand, relate every possible impacting software variabilities to the corresponding hardware change variabilities, as illustrated in Figure 3. This matrix can be configured as OCL constraints [17] to validate product instantiation in application engineering. Observe that for example automatic landing software feature requires Landing gear hardware feature.

![Fig. 3. A portion of the dependence matrix for the UAV Product Line.](image-url)

Once the refined feature management is completed, the designer can move to the partitioning of the system in question: depending upon the requirements and resources in hand, he or she can determine which part of the system needs to be implemented in hardware or software. It is possible, although it substantively increases SPL cost, to increase the number of possible SPL members derivation by implementing system features in a redundant way, i.e., whenever possible, maintain in the repository features implementations in both hardware and software. Thus, it becomes part of the Application Engineering, whenever possible, to decide if the features implementation component should be integrated in the product by software or hardware.

For a description of the different steps related to each design level by means of MARTE concepts, see the work of Quadri et al. [2], which can be adapted to this approach by adding mandatory, optional and alternative stereotypes to the target models for the hardware and software specification, architectural definition, components and subsystems design and simulation. It is also important to perform validation activities in every model to improve safety. At the end of this phase our repository contains all the artifacts and the domain engineering is concluded.

B. Application Engineering

Application Engineering corresponds to a product line target system configuration by assembling reusable artifacts from the repository. This step is the responsibility of the application
engineer, who is responsible for eliciting system requirements, verifying that the system being developed belongs to the SPL and, if so, selecting the desired features for the target system. The feature selection defines which system artifacts should make part of the new system. From this configuration, the target system is assembled from reusable components developed in the Domain Engineering. To end this step, it is necessary to conduct simulation and testing also on the target system to validate it. The software system instantiation can be automatic by configuring the SPL in an application generator like Pure::variants [11] or other available tools.

V. CONCLUSIONS

This paper aimed to present an approach for the development of critical embedded systems product lines with the use of MDD in both Domain and Application Engineering, as well as software and hardware variability management. To fulfill this objective we have used a subset of UML profiles like SysML, MARTE and a simple variability management profile combined with the feature modeling and a dependency matrix to simplify the challenges of critical embedded system product line development. The use of the proposed approach in the CES domain can bring the benefits of PL and MDD techniques like reducing system costs and time-to-market and increasing system reliability. The steps, models and concepts from the proposed approach have been applied by a UAV product line case study that was not shown due to lack of space. For future work we propose to evaluate the use of other MDD techniques during domain engineering to increase the advantages of model transformations, to facilitate the adoption of the proposed approach, and for automatic generation of target system models.

REFERENCES


75