The ProLiCES Approach to Develop Product Lines for Safety-Critical Embedded Systems and its Application to the Unmanned Aerial Vehicles Domain

Rosana T. Vaccare Braga
Kalinka R. L. J. Castelo Branco
Onofre Trindade Júnior
Paulo C. Masiero
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
São Carlos – SP – Brazil
rtvb,kalinka,otjunior,masiero@icmc.usp.br

Luciano O. Neris
AGX Tecnologia Ltda – São Carlos – SP – Brazil
luciano.neris@agx.com.br

Martin Becker
Fraunhofer IESE – Kaiserslautern – Germany
martin.becker@iese.fraunhofer.de

Abstract

This paper presents ProLiCES, an approach for the development of safety-critical embedded applications and its usage to develop a product line for unmanned aerial vehicles (UAV). The motivation of ProLiCES emerged after the development of Tiriba, a family of small, electric-powered unmanned aircraft. Most software artifacts produced for Tiriba required modifications to be reused in a more complex project, the SARVANT, which has to accommodate several new features that increase the variability of the end products. In the Tiriba project, a methodological approach, named SAFE-CRITES, was defined and used. Special care was taken about software reuse, based on Model Driven Development and automatic code generation. The certification process, based on the DO-178B standard, was also taken into account. ProLiCEs extends SAFE-CRITES to integrate Product Line Engineering into the development process, aiming better software reuse. This extension was done by creating a second parallel path to the process, dealing with the Product Line Domain Engineering. ProLiCES is being currently used in the SARVANT project, which will deliver a much more complex UAV and is estimated to be deployed in two years.

Keywords: Software Product Line, Unmanned Aerial Vehicles, Safety-Critical Systems, Certification

1 Introduction

Embedded systems are computing systems that are part of a larger system. They provide a predefined set of tasks, normally dedicated to a particular real time application, and present special requirements. In fact, they typically provide real-time monitoring and control for an entire system. These systems are considered to be safety-critical when failure events can lead to human lives losses or high valued asset losses. In some applications, such as in aviation, safety-critical embedded systems must present failure rates as low as a serious fault every 10^5 to 10^9 hours of operation.

In today's world, general-purpose computer systems (aka desktops, notebooks, etc.) are heavily outnumbered by embedded systems. Almost any household or automotive equipment includes an embedded system,

making these systems economically much more important than general-purpose systems. The firmware, ie, the software part of an embedded system, is also costly. Normally costing US$ 15-30 per line of code, in defense systems this cost is about US$100 [1].

The role of certification in the embedded systems domain should not be underestimated, principally for safety-critical embedded applications. Without certification under adequate standards, an embedded system product cannot reach a specific market. This is an important, but costly process. In highly critical applications, such as the space shuttle, this cost can reach US$1000 per line of code. A simple application in this class with 1000 lines of code can cost a million dollars from project commencement to delivery [1].

Being costly, the software development process deserves special attention. Many methodological approaches have been proposed for the development of general use applications, but only a few of them focuses on the development of embedded systems software. This paper proposes an approach that is heavily based on mechanisms for software reuse such as MDD (Model Driven Development), SPL (Software Product Line), and automatic code generation in the embedded system domain. The main goal is the reduction on the overhead costs imposed on the development of safety-critical embedded systems against the development costs of general use applications. The proposed approach is especially important in the avionics domain, in particular for unmanned aircrafts, which are complex safety-critical embedded systems. It is worth highlighting that this work corresponds to an extended version of a work presented before [2], containing more details about several process activities, including certification-related activities, as well as presenting a few more examples during the case study.

The remaining of this paper is organized as follows: Definitions and Related work are found in Section 2; Previous work on which ProLiCES is based, the Tiriba project and SAFE-CRITES, appear in Section 3. ProLiCES is described in Section 4. Section 5 presents the preliminary results from the use of ProLiCES in a very complex unmanned aircraft system, the SARVANT Project. Finally, the section 6 presents the conclusions of this work.

2 Background

2.1 Definitions

This paper proposes an approach for the development of complex embedded systems. The authors define a "complex embedded system" as the class of systems that present at least four of the following features: 1) multiprocessor or multi-core; 2) 10k+ lines of code (without comment lines); 3) multi-language; 4)RTOS based; 5) 10+ different types of I/O communication; and 6) Critical nature.

Unmanned Aerial Vehicles (UAV)s are aerial vehicles that do not need a human operator and, therefore, they can fly autonomously or be piloted remotely. Unmanned Aircraft Systems (UAS) include the aircraft and all associated elements such as the payload, the ground control station, and communications links [3].

A UAS is a typical application of a complex safety-critical embedded system. The term is adopted by both the FAA (Federal Aviation Administration) and the international academic community. A UAS can operate for a long period of time without human pilot intervention. There are different types of UAS presenting different capabilities. Some aircrafts, for example, can fly autonomously following a pre-programmed flight path, while others fly receiving commands from pilot-operated ground stations.

An SPL sounds to be a very useful technique in this context. SPL is a set of software systems that share common and managed features and fulfills requirements of a particular market segment [4]. A feature is a product characteristic that both customers and developers consider important to describe the product and to differentiate one product from another [5]. An SPL is developed from a common set of core assets in a systematic manner [4]. It can be developed from three different adoption perspectives [6]: proactive, where there is not an existing product to be analysed, so prospective investigation needs to be done to look ahead and figure out the features that would be relevant to compose the SPL; reactive, which starts with one existing product, and then evolution is done iteratively, according to customer needs or market issues; and extractive, where the SPL is extracted by analysing two or more exiting products.

Model Driven Development (MDD) is an approach for software development that focuses the importance of models in the software life cycle, considering them part of the final product [7]. The key idea is to keep models as simple as possible. Most of the complexity is dealt with by transformations that can be done automatically. This helps increasing the quality of the final product, as well as the future evolution of the software, as new changes are done directly onto the models followed by automatic code generation.

2.2 Related Work

Several works concerning the use of SPL in the embedded systems domain are reported in the literature. Nord [8] describes an architecture for embedded systems supporting SPL. His work proposes some quality

goals and the means to achieve these goals. For example, portability is achieved by a layered design and libraries. To ease the inclusion and configuration of features, he suggests an approach based on a set of rules. Real time requirements are provided by a task-based architecture, making the task management more flexible.

Yoshimura et al. [9] apply SPL on existing embedded systems. Their approach checks the economical feasibility of an SPL, analysing the return of investment and the effort required to develop reusable artifacts. The analysis of commonalities is focused on the source codes of existing embedded systems. The study does not take into consideration specific real time requirements.

Koong at al. [10] present an approach where the concept of SPL was used to ease dynamic reconfiguration of a product line for Lego robots. They use XML configuration files to describe the composition of the system. When a hardware or software functionality is created or suffers changes, the XML configuration file changes reflecting the new configuration.

Habli [11] defines and evaluates a model-based approach to assure systems and processes in a safety-critical product line. A safety metamodel is included in the approach, allowing the developed artifacts to be reused in a traceable and justifiable way and the impact analysis on the validity of the entire system.

Polzer et al. [12] present a model-based SPL engineering process based on a Rapid-Control-Prototyping system. Embedded controllers are modeled in Simulink and code is generated automatically. They introduce a Hardware Abstraction Layer (HAL), which allows to exchange and adapt hardware components without changing the model-based implementation. Their approach is illustrated with a parking assistant application, tested on a experimental vehicle performing automatic parking maneuvers. Products are obtained with the support of pure::variants [13].

This paper proposal is more aligned with the work of Polzer [12], covering both domain and application engineering. Additionally, there is special concern with the validation steps necessary to fulfill certification requirements to reduce costs by reusing certified artifacts.

3 Previous Work

3.1 The Tiriba Project

Tiriba is the name of a family of small, electric-powered unmanned aircraft developed for monitoring applications in agriculture and security. Despite its apparently simplicity, as can be seen in Table 1, the avionics of Tiriba is a complex safety-critical embedded system. It supports autonomous missions, has a ground station, smartphone based, with endurance of 40min/1h30min. Payload of 0.7 kg is supported. The hardware has a mainboard with four networked processors that are allocated to the four main blocks of the system: mission controller, flight controller, pressure unit and inertial unit.

Table 1: Tiriba - Basic Specifications

Propulsion	Electric,1.2KW
Max Takeoff weight	3 Kg
Payload	0.7 Kg
Endurance	40min/1h30min
Cruisier speed	100Km/h/60Km/h
Takeoff	Hand launch/catapult
Landing	Automatic, parachute
Missions	Autonomous
Ground Station	Smartphone based
Assembly time	10 min

3.2 SAFE CRITES

Since the beginning of the Tiriba project, special attention was given to techniques that could improve software reuse. The main aspect was the use of MDD with automatic code generation. Among several tools for system modeling and simulation, Matlab Simulink [14] was chosen. This tool is particularly adequated considering the availability of specific application domain modules. To further increase reuse and to make easier future certification efforts, a methodological approach, named SAFE-CRITES [15] was proposed and applied. Figure 1 illustrates its main steps, which are based on the following statements:

- Life cycle based on a sequence of steps separated by rigorous validation effort, with an extra effort to avoid the need of revising a previous phase, as occurs in the iterative or spiral development approach, commonly used in conventional system development;

- Maximize the use of modeling, simulation, and automatic code generation tools, aiming shorter development times, fewer errors and easier documentation and certification of the generated code;
- Maximize code reuse of previously tested and certified components of product families. A new product member should be, most of the time, obtained by the reorganization of components and their mappings to new hardware architectures;
- Not imposing (although not forbidding) the use of object-oriented programming, as some standards do not contemplate this technology. Despite OO advantages, its acceptance in the embedded systems domain is not always unquestionable, mainly because the extra cost regarding cycles and memory area designated to program and data, as pointed out in several researches [16, 17].

The total development time predicted for the project was 6 months, including hardware/software development, resulting in a combined effort of 2600 working-hours of a multi-disciplinary team including computer science engineers, electronic engineers, aeronautic engineers, electronic technicians, composite materials technicians and unmanned aircraft field operators (the aircraft development took itself 500 working-hours). In the end, the entire system has 25.000+ lines of C code, excluding comment lines. The development effort was by far exceeded for several reasons, including unexpected first-time-use training difficulties for the tools and adopted methodological approach; loss of the team leader, leaving a serious lack of application-domain knowledge; non-predicted necessity of bug bypasses on the basic tools; system requirements changes during development; and unrealistic effort estimation.

4 ProLiCES - Product Line on Critical Embedded Systems

4.1 Further Motivation for Extending SAFE CRITES to SPL

At the end of the Tiriba project, an evaluation was carried out to see in which extent software reuse would be possible, as the hardware for the new project, the SARVANT, had changed completely to deal with new requirements. Although they can be reused after some adjustments, the artifacts produced in the Tiriba project do not fully comply with the requirements of the wider domain represented by the SARVANT family. This is where SPL Engineering can help and this was the main motivation behind ProLiCES.

4.2 ProLiCES Overview

ProLiCES is an extension of SAFE-CRITES, based on product line software engineering, to support the development of safety-critical embedded systems. An overview of ProLiCES activities is shown in Figure 2. A two-path parallel life cycle is proposed, where Domain Engineering and Application Engineering can be conducted simultaneously or separately, depending on the current scenario faced by the software enterprise. This approach allows the SPL to be developed from any of the three adoption perspectives mentioned in Section 2.1: proactive or extractive (Domain Engineering first) or reactive (Application Engineering first). If there is an immediate demand for a real-world product, the software organization can follow the Application Engineering path, possibly reusing previously tested artifacts from a previous SPL repository. On the other hand, if the schedule allows investment to build the SPL, then the Domain Engineering path can be followed.

It is worth noting that ProLiCES is highly concerned with validation of the several artifacts produced during the development, as they are essential to guarantee a high quality product at the end of the process, which should be later certified under pertinent standards. To highlight this characteristic of ProLiCES, validation activities are presented in separate lanes. Each validation activity is linked to a development activity, and it is performed in parallel or shortly after the activity is finished. Process iterations are not included intentionally in Figure 2, as it would make the figure much more complex to understand.Iterations are essential in ProLiCES. At any time during the execution of activities, it is possible to return to previous activities, whenever errors are found and need to be fixed.

4.3 Domain Engineering

The main goal of Domain Engineering is to produce reusable artifacts that can be easily composed based on SPL concepts: a set of core assets that will be present in any SPL product, as well as a set of variable assets that are included only if specific requirements must be fulfilled by an specific application.

The first step of Domain engineering is to perform a planning of the development activities, 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 the second step is carried out, where a domain analysis is done to obtain a document that describes functional and non-functional requirements for SPL products. It is out of the

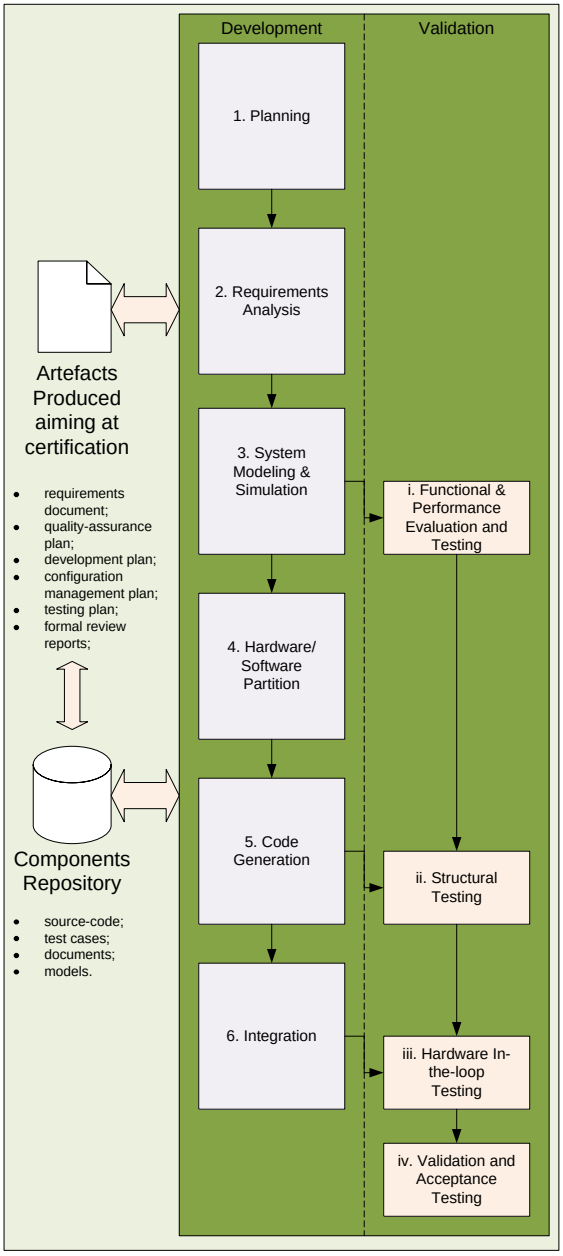

Figure 1: SAFE-CRITES Approach [15]

scope of this work to propose domain analysis techniques, as they can be easily found in the literature. For a survey on this topic, see the work of Prieto-Diaz and Arango [18]. In ProLiCES, it is important that the domain analysis result in a requirements document where there is a distinction between functional and non-functional requirements, including mandatory, optional and alternative requirements. This document is inspected by an expert team, together with potential customers of the SPL. Refinements can be done as long as necessary to produce a validated artifact that can be included in the SPL repository. Validation should conform to certification rules to assure that the product can be included. Later on, when concrete products are instantiated, this document will be matched against the specific product requirements. Again, as this is an iterative and a two-path process, if concrete products have extra requirements not covered by the SPL, the requirements document can be further reviewed to include these new requirements.

The third step is feature modeling, which is based on the requirements document produced during domain analysis. Considering that a complete requirements document was produced and validated in the previous activity, this activity will be much easier to execute. But it is important to notice that some organizations will prefer to begin with feature modeling to have a big picture of the system features, and then produce the requirements document. In this case, the feature model will be used as basis to write the requirements document.

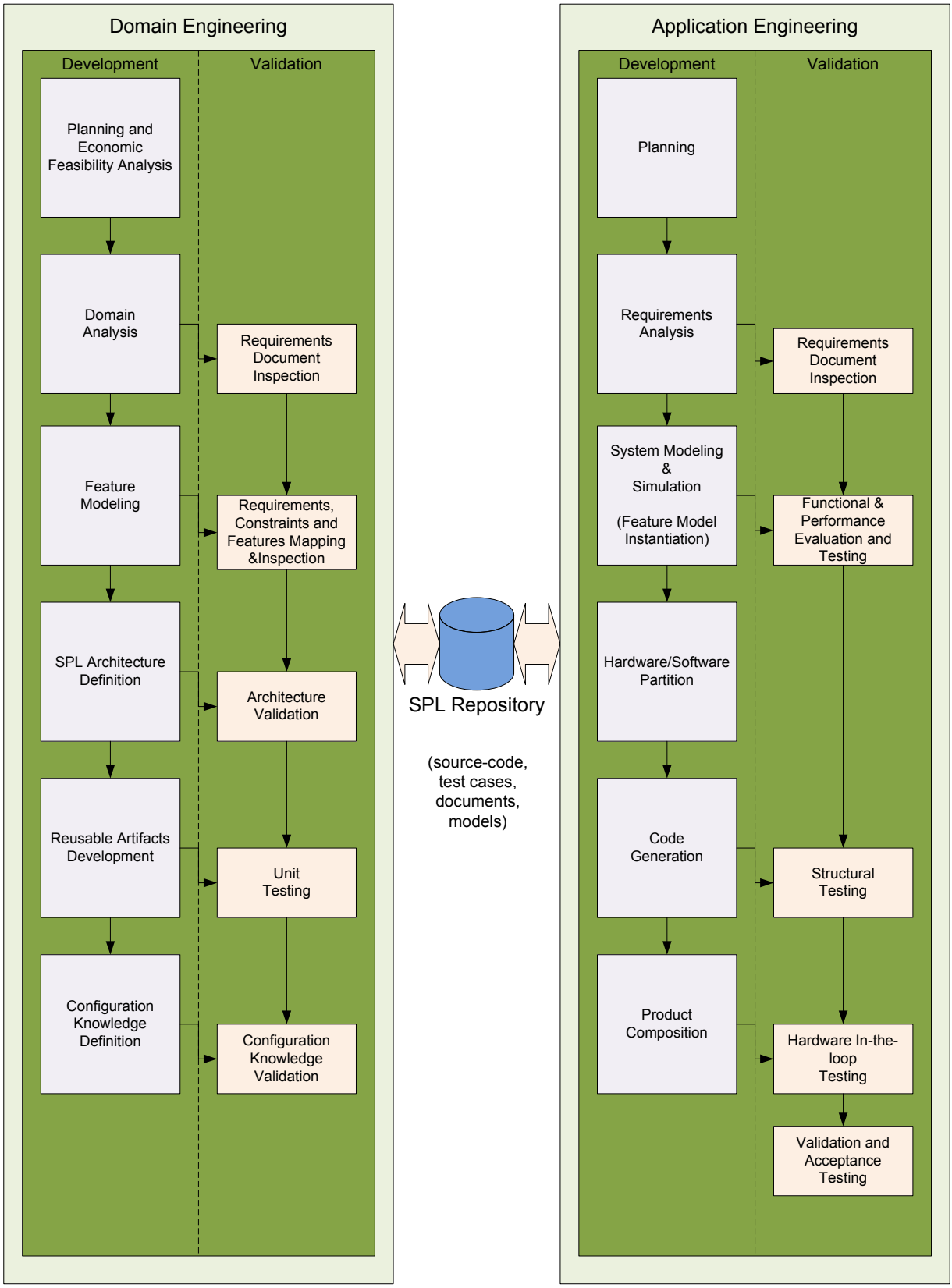

Figure 2: An Overview of ProLiCES

The resulting feature model contains mandatory, optional, and alternative features, as well as the constraints that must be valid during product instantiation. A constraint can represent dependencies among features, as for example inclusion (feature A requires feature B) or exclusion (feature A excludes feature B). The validation of this activity is done by inspecting the feature model against the requirements document and checking the consistency among constraints. A requirement can originate one or more features. Every requirement should be covered by at least one feature. But a feature can refer to one or more requirements. The feature model is also stored in the SPL repository and it will be the most important artifact to ease the instantiation of products during application engineering.

It is important to observe that both hardware and software features are included in the feature model. The decision about whether they will be implemented in hardware or in software will be made later, in the Application Engineering path, possibly depending on the result of the performance tests.

The next step is to define the SPL architecture, which involves the identification of the main components that will be built and the way how they relate to each other. The architecture can be designed in several levels. A high level architecture gives an overview of the SPL, while the next levels give more details about each component and the interfaces between them. Again, a validation of this architecture is very important to guarantee that no requirements or features were forgotten, and that components and interfaces are correct and consistent.

Having validated the SPL architecture, an analysis is done to verify which artifacts can be developed and stored in the SPL repository. artifacts corresponding to mandatory features are usually easier to develop, as the source-code is fixed and used as it is. On the other hand, artifacts that implement optional and alternative features are more complex to design, as their integration with other fixed artifacts requires a flexible interface. Unit testing is done at this point to validate isolated artifacts. Test cases used in this activity are stored in the SPL repository to be reused during application engineering with real-world products.

Finally, a very important activity to be carried out during domain engineering is the definition of the configuration knowledge needed to instantiate products. In fact, this activity can start as soon as artifacts are produced, analysing their relation to the features, allowing to reason earlier about non-realised features or support change impact analysis. Basically, the configuration knowledge consists of mapping each feature of the SPL feature model to one or more artifacts that implement it. It is also important to implement the constraints among features and guarantee that the application engineer chooses the correct features, avoiding conflicts that can infringe the constraints. A validation of the configuration is needed and it can be done automatically by tools if available, as for example pure::variants.

4.4 Application Engineering

Application engineering aims at the production of real-world SPL products. As ProLiCES allows a reactive SPL adoption perspective, application engineering would be conducted even if an SPL repository does not exist yet, i.e., a first product could be developed and, afterwards, its reusable parts reorganized to begin the construction of a repository. The Tiriba project followed this path. In fact, the application engineering path in Figure 2 is basically SAFE-CRITES with minor modifications.

Application engineering begins with software planning, where the project is planned following typical project management guidelines. Several artifacts are expected in this step, particularly when certification under DO 178-B is desired, as it demands the production of a number of planning documents: plan for certification, plan for software development, plan for software verification, plan for software configuration management, and plan for software quality assurance. All these artifacts can be partially reused from previous developments, adjusting them to the specific project.

The requirements analysis for the product is the next step, where specific needs for the concrete product are established. Likewise to Domain Engineering, the requirements document produced in this step must be inspected by experts to guarantee its correctness. If an SPL repository already exists, the SPL requirements document can be the basis for the product requirements document. This is certainly easier than starting from scratch, since functional and non-functional requirements are already defined and pertinent requirements to the target product can be selected.

The next step is to model and simulate the system. Here there are different strategies to follow, depending on whether it is the first product being developed, or the SPL already exists and is being evolved. In the first case, the features of the product under development are identified and can be considered the first feature model, where all features are mandatory. In the second case, domain engineering has been already done in the past, so the application engineer has only to instantiate the feature model by selecting the desired features that match the product requirements. New requirements or features should be analysed and included, if appropriate, in the SPL repository. This can be a complex task, as the integration of new functionalities is not always straightforward.

If a SPL does not exist yet, after feature modeling it is necessary to define the system architecture and to develop the reusable assets, and this can be done by following the same guidelines used for domain engineering (see Section 4.3). On the other hand, when the SPL already exists, usually it is enough to select the desired features and reuse the corresponding implementation assets. Eventually, when the SPL does not cover 100% of the new product requirements, new features will need to be incorporated into the architecture and corresponding assets need to be developed as well.

Having completed the modeling, simulation can be carried out and adjustements can be done in the models until the expected results are achieved. Functional testing can be done at this point to check if the feature model instance attends the product requirements.

After choosing the features that will be part of the product, the application engineering has to decide which features will be implemented by software or by hardware. This step is called Hardware/software partition. Performance simulations done in the previous step can help this decision, and several iterations may be needed to find the best solution for the particular product. Provided that an adequate division is achieved, code can be generated for the chosen features. Some of these features can be represented by models, such as state machines or dataflow diagrams, while others can have an associated source or pseudo code. Structural testing can be done at this point using appropriate testing criteria.

A HAL layer is necessary to interface the automatically-generated code to the target hardware. The development of this layer must be done every time a block of hardware changes. Notwithstanding, hardware can be semi-automatically generated making use of code generators for hardware description languages, such as Verilog Hardware Description Language (VHDL). Since integrated, hardware and software of the new product can be submitted to hardware in the loop testing. This is an useful testing technique for safety-critical embedded systems, as the entire system can be tested without the risk of a first-time real-world operation.

Finally, product integration takes part, where the embedded system is integrated to the final system and is ready for final validation and acceptance tests.

4.5 The Role of Certification Standards

Certification, under pertinent standards, is an important phase in the development of an embedded system, even more if it is a safety-critical embedded system. Standards exist for a multitude of reasons, not only to assure product quality. Economic interests and political aspects are also important. In this paper it is discussed avionics systems, covered by standards such as DO-178B [19] (and its recently issued version DO-178C) and DO-254. In ProLiCES, the authors intend to support another important application domain: the automotive, where standards like MISRA C and ISO 26262 have great importance. Although there are specific requirements, all standards try to guarantee the quality of products, normally ruling over the development process, heavy documentation and testing.

The main standard for avionics software is the DO-178B [19]. It allows certification in several levels (A, B, C, D, or E). “A” is the most difficult level to achieve, and is recommended when the consequences of a potential software failure are catastrophic. “E” is the easiest level, used when there are no dangerous effects if the software fails. The effort to obtain an “A” level is greater and more expensive to the software development enterprize when compared to the effort to get a lower level.

Certification is an important subject for ProLiCES. In this direction, certification standards have been studied and their specific requirements incorporated to the development process, mainly concerning documentation and test cases generation. This is out of the scope of this paper, but we provide a brief summary to illustrate typical activities that need to be carried out during development aiming at obtaining product certification.

DO-178B [19] supplies objectives to be met during all processes of software development, from software planning, passing through analysis, design, coding and testing. For example, during the software requirements process a goal to be achieved is that “software high-level requirements comply with system requirements” and the output of this process should include “software verification results” where this compliance can be evidenced. Notice that there is no specification of how to achieve this objective. In ProLiCES, we have defined, for each application engineering phase (see Figure 2), a set of activities and corresponding artifacts that should be followed aiming at certification, as exemplified in Figure 3. For the previously mentioned goal, for example, we have established requirements activities (system and software analysis) including artifacts that help ensuring that each software requirement has an associated system requirements related to it and vice-versa.

As future work we plan to associate weights to artifacts according to each certification level. These weights will be obtained empirically, and will be adjusted at each new development. Our focus is presently on application engineering, because DO-178B is not concerned with SPL certification, but only with product certification. It is important to notice that several verification and validation activities are carried out

Step	Activity	Artefacts
Planning	Planning activities	Plan for certification
		Software Develop. Plan
		Software Verification Plan
		Soft. Config. Management Plan
		Soft. Quality Assurance Plan
Requirements analysis	System Analysis	System Requirements Document
	Syst. Req. Doct. Inspection	System Req. Verification Results
	Software Analysis	Software Requirements Document
	Soft. Req. Doct. Inspection	Soft. Req. Verification Results
	Final Requirements Inspection	Mapping between syst x soft
System Modeling & Simulation	Instantiation of Feature Model	Feature Diagram Instances
		Instances of Dependencies among features
	Feature Model Instance Inspection	FM Verification results
	Simulation	Simulation Results
	Evaluation and Analysis of simulation	Analysis Results

Figure 3: Partial Example of Mapping ProLiCES application engineering phases to certification activities/artifacts

Table 2: SARVANT - Basic Specifications

Propulsion	AVGas, 32KW
Max Takeoff weight	120 Kg
Payload	45 Kg
Endurance	30.5 hours
Cruisier speed	200Km/h
Takeoff	Conventional/On a vehicle
Landing	Conventional on ski
Missions	Autonomous
Ground Station	Network of workstations
Assembly time	3 hours

during domain engineering, in order to produce high quality SPL assets that will be easier to certifify further, although specific activities regarding certification are assigned only during application engineering.

5 Preliminary Results

5.1 The SARVANT Project

ProLiCES is being used in the SARVANT project, where the development of a long-endurance unmanned aircraft with the specific mission of carrying a dual band SAR sensor (Synthetic Aperture Radar) is being conducted. It is a jointly effort of AGX Technology Ltd (www.agx.com.br) and Orbisat Inc (www.orbisat.com.br), having the INCT-SEC (www.inct-sec.org) as the academic branch and FINEP (www.finep.gov.br) as the main sponsor. From the user's perspective, SARVANT is a different and more complex product than Tiriba, as can be observed in Table 2. From the development perspective, the Tiriba SPL can be extended to cover the new features of the product SARVANT, that also can be viewed as a family of similar aircrafts.

Although not visible to users, several features of the SARVANT family are originated from the Tiriba family. All these new features change radically the variability level of products of the SARVANT family, according to different possible configurations. Several examples of new features are described next to illustrate these variabilities.

The first new feature is the MOSA concept (Mission Oriented Sensor Array), the corresponding interface (SSI - Smart Sensor Interface) and protocol (SSP - Smart Sensor Protocol) [20]. A MOSA unit, at the system startup, will conduct a plug-and-play negotiation with the aircraft to agree on a range of flight parameters (performance, among others), to evaluate the feasibility of the mission from fully feasible through partially/degraded feasible to not feasible. In flight mode, the MOSA unit controls the aircraft during the mission.

The second new feature is redundancy of software and hardware. Ideally, to avoid systematic bugs in highly critical embedded systems, both software and hardware redundant units must have different development cycles and different development teams. This latter issue has not been considered in the SARVANT family. In the first attempt to implement redundancy in the SARVANT family, the hardware will be focused. Organized as a redundant computer network, the SARVANT avionics can have multiple units providing the same functionality in a redundant way. For example, multiple flight controllers can provide similar data to multiple control-surface controllers that evaluates the correct value to use. A further explanation on this feature is beyond the scope of this paper and it will be published in the near future.

The third new feature is IFA (In-Flight-Awareness). The key idea is to provide the aircraft with sensors and heuristics to replace the capabilities of the missing human pilot. Smoke and vibration sensors are considered. DTM (Digital Terrain Model) and Geopolitical information are also important. This feature will allow the aircraft to have a more adequate behavior in some circumstances. For instance, in a faulty condition, the aircraft itself can judge the best place for an emergence landing based on the IFA database and some heuristics.

The application of MDD allows the reuse of most artifacts (at model level), previously developed for the Tiriba project. At the time of this writing, the SARVANT project has already done several activities of the domain engineering, such as planning and economic feasibility analysis, domain analysis, feature modeling, SPL architecture definition, reusable artifacts development, and configuration knowledge definition (these two last are both ongoing work, byproducts of the Tiriba project).

To better understand the new concepts and features of the SARVANT family, an architectural view has been proposed [20] as shown in Figure 4. It is possible to observe that many components are especially designed to provide the new features mentioned before. For example, to improve reliability through redundancy, a number of critical components have been replicated. Components that implement IFA and MOSA were also included.

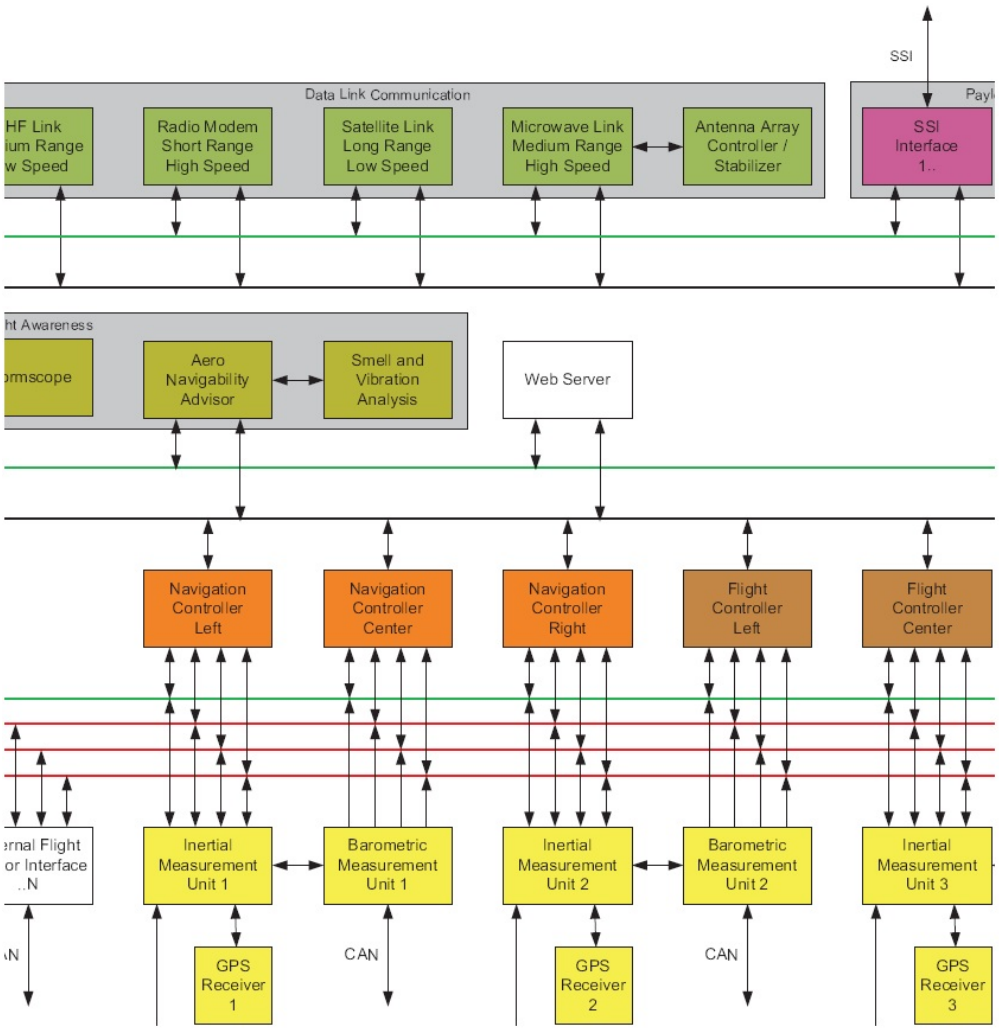

Figure 4: Architecture of the SARVANT Family (partial view)

The application engineering has begun, with planning, requirements analysis already done, and the other phases beginning now. Figure 5 shows part of the requirements document for the SPL, where stereotypes have been used to highlight mandatory, optional or alternative requirements. The hardware engineering is under design, with the aircraft designed and prototype almost finished.

OE4– *«a»* The UAV payload can vary according to the specific mission. The possible types are: photographic cameras, video cameras, altitude (sonar ou lidar based), MOSA (Mission Oriented Sensor Array) subsystems.
OE5 – *«a»* The system should allow hand launching or catapult launching.
OE6 – *«a»* The system communication is done via USB and, optionally, via bluetooth, or by radio modem and/or by a video transmitter (this could be analog VHF or digital)

4. Functional Requirements

F1 – *«m»* The system should allow the definition of missions composed of waypoints and actions.
F1.1 – *«m»* A waypoint is composed of a latitude, a longitude and an altitude
F1.2 – *«m»* Sweep missions following a grid, should be possible.
F1.3 – *«m»* A mission should be assigned an identifier and a name to ease its retrieval.
F2 – *«m»* Waypoint accuracy can be pre-specified by the user.
F3 – *«m»* The system must interpret the mission and perform the calculations and maneuvers necessary to adequately (within the specified accuracy) reach the waypoints.
F4 – *«a»* The actions associated to a mission can be: taking pictures of the region under flight using photographic cameras, and/or broadcasting live images using video cameras. It is possible to broadcast and take pictures at the same time.
F5 – *«m»* The photographic or video camera can be turned off during the flight through a command issued by the user at the ground control station.
F6 – *«o»* The camera subsystem should compensate for the banking of the aircraft allowing better vertical images.

Figure 5: Requirements document (partial view)

5.2 A Feature Model for Fixed Wing UAVs

A broader feature model was partially developed, having the feature model of Tiriba as a start point, encompassing not only the SARVANT family, but almost any fixed-wing unmanned aircraft configuration. A part of this model can be seen on Figure 6.

It is expected the SARVANT project to be ten times bigger than the Tiriba project, resulting in 250.000+ lines of automatically (most part) generated code. ProLiCES is the best bet of the authors to undertake this class of development effort.

The SARVANT SPL is being developed using ProLiCES in an incremental way. This means that several iterations are planned to occur, starting with the development of the most usual features to fulfill the requirements of great part of the applications demanded so far, and then other features will be incorporated as needed. The reactive approach to SPL was used, as we have started from an existing product (Tiriba) and new features are being incorporated towards a bigger SPL.

6 Conclusion

ProLiCES has been proposed to solve some practical problems found in the development of the Tiriba project, mainly related to software reuse. At its end, Tiriba resulted in 25.000+ lines of code, split onto four networked processors.

The development of the SARVANT, a family of complex unmanned aircraft, has been started using ProLiCES. Its is expected for SARVANT ten times more lines of code spread over tens of processor boards (depending on the amount of features and level of redundancy). Software reuse is imperative, starting with the artifacts produced in the Tiriba project.

At the time of this writing, current work is focused on the Feature Model for the SARVANT Family (under revision and checking) and the redefinition and mapping of software artifacts developed for Tiriba according. As a big project, the SARVANT can take up to two years more to complete. During this time, ProLiCES will be evaluated and eventually slightly modified to attend other safety-critical embedded applications, such as automotive electronics.

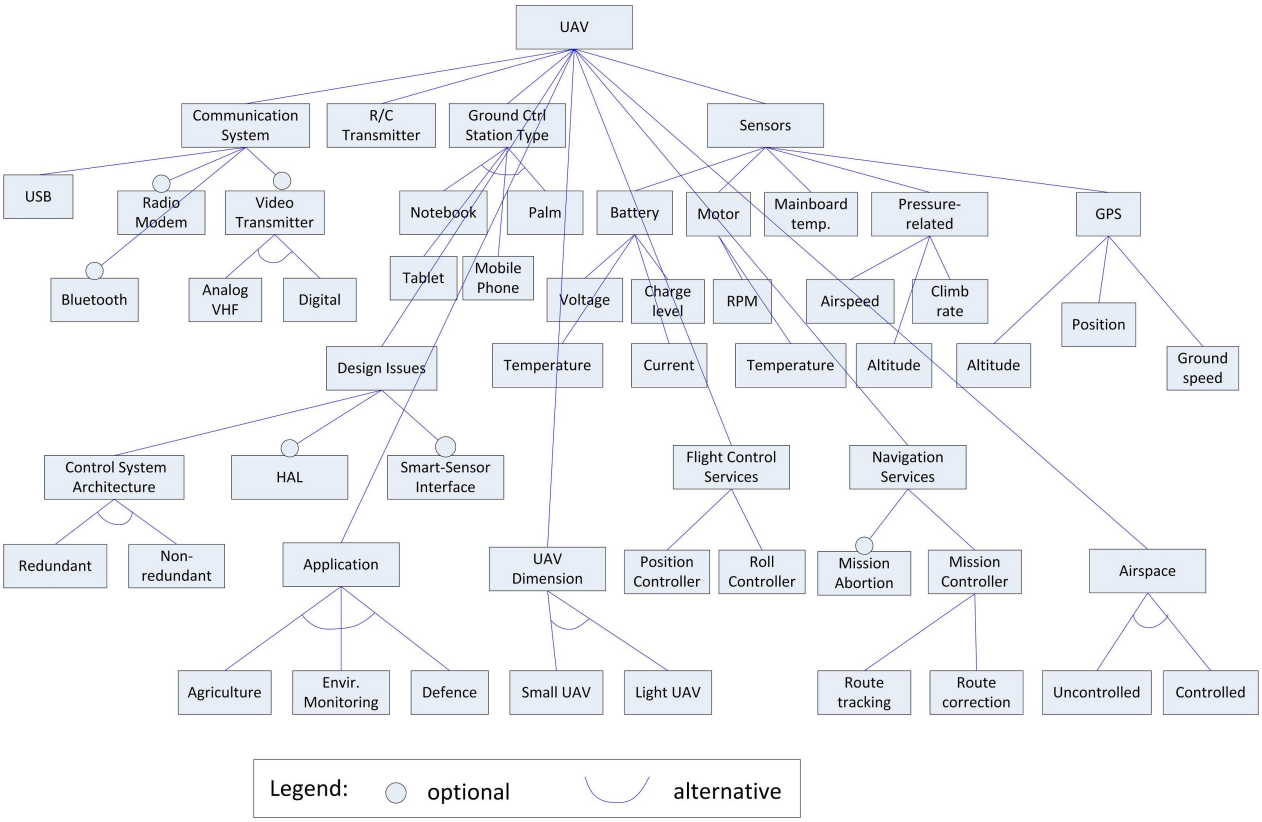

Figure 6: Feature Model (partial view)

Certification under the DO-178B is also a goal for the SARVANT project. ProLiCES must provide facilities for this task concerning easier documentation and easier test cases generation. This is an ongoing parallel work that may have impact on the process. Another key point in ProLiCES, not detailed in this paper, is a third path of the process, the Hardware Application Engineering, started after the hardware/software partitioning. As future work, automatic hardware generation can be used, although it is not applicable for the SARVANT project.

Acknowledgments

The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology - safety-critical Embedded Systems - Brazil), processes 573963/2008-9 and 08/57870-9.

References

[1] RTO, "Validation, Verification and Certification of Embedded Systems, TR-IST-027," NATO, October 2005, Tech. Rep., 2005.

[2] R. Braga, K. R. L. J. C. Branco, O. J. Trindade, P. C. Masiero, L. O. Neris, and M. Becker, "ProLiCES: An approach to develop Product Lines for Safety-Critical Embedded Systems," in *Proceedings of CLEI - XXXVII Latin-American Informatics Conference*, 2011, pp. 991 – 1006.

[3] GAO, "Unmanned aircraft systems - federal actions needed to ensure safety and expand their potential uses within the national airspace system, GAO-08-511," GAO, 2008, Tech. Rep., 2008.

[4] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman, Aug. 2001.

[5] M. L. Griss, "Implementing product-line features with component reuse," in *6th International Conference on Software Reuse (ICSR)*, 2000, pp. 137–152.

[6] C. Krueger, "Easing the transition to software mass customization," in *Proceedings of the 4th International Workshop on Software Product-Family Engineering*, 2001, pp. 282–293.

[7] M. Völter and I. Groher, "Product line implementation using aspect-oriented and model-driven software development." in *SPLC*. IEEE Computer Society, 2007, pp. 233–242. [Online]. Available: http://dblp.uni-trier.de/db/conf/splc/splc2007.html#VolterG07

[8] R. L. Nord, "Meeting the product line goals for an embedded real-time system," in *Software Architectures for Product Families*, ser. Lecture Notes in Computer Science, F. van der Linden, Ed. Springer Berlin / Heidelberg, 2000, vol. 1951, pp. 19–29.

[9] K. Yoshimura, D. Ganesan, and D. Muthig, "Defining a strategy to introduce a software product line using existing embedded systems," in *Proceedings of the 6th ACM/IEEE International conference on Embedded software*, 2006, pp. 63–72.

[10] C.-S. Koong, H.-J. Lai, and K.-C. Lai, "An embedded software architecture for robot with variable structures," in *International Conference on Frontier of Computer Science and Technology*, 2009, pp. 478–484.

[11] I. M. Habli, "Model-based assurance of safety-critical product lines," PhD Thesis, University of York, York - UK, Sep. 2009.

[12] A. Polzer, S. Kowalewski, and G. Botterweck, "Applying software product line techniques in model-based embedded systems engineering," in *Proceedings of the Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2009), at the 31st International Conference on Software Engineering*, 2009, pp. 2–10.

[13] Pure-Systems, "Pure::variants," 2011, available on 2011, May 30 at: http://www.pure-systems.com/pure variants.49.0.html.

[14] Mathworks, "Simulink - simulation and model-based design," may 2011, available on 2011, May 30 at http://www.mathworks.com/products/simulink/.

[15] R. Braga, K. R. L. J. C. Branco, L. O. Neris, and O. J. Trindade, "Safe-crites: Developing safety-critical embedded systems supported by reuse techniques," in *Proceedings of IRI - International Conference on Reuse and Integration*, 2011, pp. 206–211.

[16] A. Chatzigeorgiou and G. Stephanides, "Evaluating Performance and Power of Object-Oriented Vs. Procedural Programming in Embedded Processors," in *Reliable Software Technologies-ADA-Europe 2002: 7th Ada-Europe International Conference on Reliable Software Technologies, Vienna, Austria, June 17-21, 2002: Proceedings*. Springer, 2002, p. 65.

[17] S. Bhakthavatsalam and S. Edwards, "Applying object-oriented techniques in embedded software design," in *CPES 2002 Power Electronics Seminar and NSF/Industry Annual Review*, 2002.

[18] R. Prieto-Diaz and G. Arango, *Domain Analysis and Software System Modeling*. IEEE Computer Science Press Tutorial, 1991.

[19] I. RTCA, "Final Report for Clarification of DO-178B – Software Considerations in Airborne Systems and Equipment Certification." RTCA/DO-248B, 12 October 2001, Tech. Rep., 2001.

[20] D. Rodrigues, E. J.C., V. M., C. M., C. J. J.B., B. K.R.C, and T. J. O., "Service-oriented architectures for complex safety-critical embedded systems: A case study on UAVs," in *I Brazilian Conference on Critical-Embedded Systems (CBSEC), São Carlos, Brazil*, May 2011, pp. 130–130.