Chapter 2

The On-board Software Reference Architecture (OSRA)

2.1. Introduction

2.1.1. Background

Space industry has recognized already for quite some time the need to raise the level of standardisation in the avionics system in order to increase the efficiency and reduce cost and schedule in the development. The implementation of such a vision is expected to | Ref! provide benefits for all the stake-holders in the space community:

Customer Agencies Significant drop in the project development lifecycle and the risk involved in the software development

what does drop of lifecycle mean? drop what?

System Integrators There would be increased competition amongst who? them to deliver at lower price and maintain shorter time-to-market and there would also be multi-supplier option

Supplier Industry Benefits from diversified customer bases and the supplied building blocks would be compatible with prime architectures across the board ? which are ?

Similar initiatives have already been taken across various industries and eg. AUTOSAR Ref! for the automotive industry is worthy mentioning. Space can benefit from these examples by studies related to how these or similar initiatives were successfully conducted and how w/ they fared. ALthough the business model is different in the automotive and the space sectors, AUTOSAR demonstrates that the need fr standardization is ~~the~~ key irrespective of the sector and is driven by the need of the industry to become more competitive.

W!

Space primes and on-board software companies have made significant progress and have implemented and/or are implementing reuse on the basis of company's internal software refernce architectures and building blocks. However, ~~in~~ for this standardization to provide maximum benefits, it has to be tackled at the European level rather than at company level.

sounds like Marketing and not like a thesis. Is this from some brochure?

not explained

Satz

ESA through its two parallel activities aimed at increasing the software reuse in on-board softwares (CORDET and Domeng) have confirmed that interface standardization allows to efficiently compose the software on the basis of existing and mature building blocks.

To refer to all ongoing initiatives and to provide a platform for technical discussions, related to the vision of avionics development through maximizing reuse and standardization, a "Space Avionics Open Interface Architecture" Advisory Group (SAVOIR Advisory Group) was created. SAVIOR Advisory Group decided to spawn a specific subgroup on-board software reference architectures called "SAVOIR Fair Architecture and Interface Reference Elaboration" working group (SAVOIR FAIRE). OSRA is the result of the R&D activities of this group.

to focus on

Ref!

The On-board software reference architecture (OSRA) is designed to be a single, common and agreed framework for the definition of the on-board software (OBSW) of the future European Space Agency (ESA) missions. It is based on solid scientific foundations and accompanied by a development methodology and architectural practices that fit the domain. A single software system would thus be an "instantiation" of the reference architecture to specific mission needs.

Ref!

The software architecture is the key to create "good quality" software because it promotes architectural best practices and contributes to the quality of the software. A bad architecture hinders the fulfillment of functional, behavioral, non-functional and life-cycle requirements. Elevating a software architecture to a software reference architecture permits to gather and re-use lessons learned and architectural best practices, give new projects a consolidated running start and promote a product line approach.

2.2. Need for reference architecture

2.2.1. Motivation

The schedules of space projects are always decreasing and the team needs to increase their efficiency and cost-effectiveness in the development process of on-board avionics. But the on-board software is getting more complex because of the trend towards more functionality being implemented by the on-board software. Therefore the overall objective of the space industry is now to standardize the avionics systems and therefore the on-board software.

Redundant to previous section

A building block approach is one of the ways to tackle this problem. In this approach, the on-board software is implemented from a set of pre-developed and fully compatible building blocks, plus specific adaptations and "missionisation" according to specific mission requirements. The target missions are the core ESA missions, ie. high reliability and availability spacecraft driven systems (eg. operational missions, science missions).

too strong

The "right" building blocks need to be produced and supplied by the suppliers to any system integrator and to achieve this, reference architectures need to be defined.

this term should be introduced based on std.-arch.-literature

And unusually?

Usually a software building block: Ref.!

- Has a clear, well defined, specified, documented function and open external interfaces for the purpose of interaction
- Meets defined performance, operation, and other requirements
- Is self-contained so that they can be used at higher-integration levels eg. board, equipment, subsystem
- Has a quality level that can be assessed
- Is applicable in well defined physical and hardware environments
- Is worth developing as they are going to be used in a bulk of ESA missions
- Is designed for reuse in different projects, by different users under different environments
- Can be made available off-the-shelf, ready for deployment under different conditions.

Separation of the application aspects from the general-purpose data processing aspects is the key to generic/reusable software architectures. The lower layers of the architectures usually handle the implementation of communication, real time capabilities, etc., and the higher level layers usually deal with the application aspects. However, there have to be ways to annotate the application building blocks (ABB) with sufficient information regarding requirements related to communication, real-time, dependability, etc., so that the platform building blocks (PBB) can provide ~~the~~ a suitable complete implementation. Development of interface specifications with reference architectures as the basis allows the implementation of the famous AUTOSAR concept: "Cooperate on standards, compete on implementation"

Ref.!

2.2.2. Software reference architecture

A ~~The~~ reference architecture is made up of two main parts: Ref.!

- An software architectural concept addressing the pure software architectural related issues
- Architectural building blocks related to functional aspects and the corresponding interface definitions which express functions derived from the analysis of the functional chains of the core on-board software domain

2.2.3. User needs

Ref.! These are some of the needs that were assimilated to guide the development of the software reference architecture:

Shorter software development time The software development time ~~schedule~~ should be ?? reduced because usually the definition of the software requirements is done at a later stage and the final version of the software is expected to be released earlier. Even though the cost of the software itself is a minor fraction of the cost of the whole system in space industry, the impact of delays in availability of the software may have a huge impact on the overall schedule and consequently on the cost of the project,

Reduce recurring costs It is important to identify and reduce ~~the~~ recurring costs and in turn help to use the project resources to focus on value added to the product or to reduce the cost of development while providing the same set of functions. Examples for recurring costs include device drivers, real-time operating systems, providing communication services, etc. and it is important to note that these cots drivers do not provide an added value and are not mission specific.

Quality of the product The level of ~~the~~ quality (timing predictability, dependability of the software, etc.) of the software must at least be the same as the one of OBSW developed with current approaches. which are?

Increase cost-efficiency Cost-efficiency is the "value" of the software product that is developed with a certain amount of budget. An increased cost-efficiency is achieved by developing the same set of functions for less budget, developing the same set of functions with more stringent requirements for the same budget and increasing the number of realized functions for the same budget. The budget available for the software development is not expected to grow and it may ~~be~~ indeed be subjected to reduction and hence new development approaches may be required to fulfill this user need. On contrary the performance of the application building blocks, eg. accuracy of the AOCS controls, is expected to grow and new complex functionalities are expected to develop.

Reduce Verfication and Validation effort The main contributor to the cost of software development are V&V activities which contribute anywhere between 50% to 70% of the overall cost. Adoption of the principle of ~~Correction~~ Correctness by Construction (C-by-C) which is one of the founding principles of choice (refer Chapter 2), analysis at early design stage and provisions for re-usability of (functional) tests are expected to reduce the Verification and Validation efforts. This also leads to shorter software development times and reduced costs. this is the one i am reading?!

Mitigate the impact of late requirement definition or change Late refinement of a system's design, evolution of the operational level, late finalization of the system FDIR, software modifications to compensate ~~the~~ problems found in the hardware ~~found~~ during system integration may often lead to the definition of new requirements or their changes for the software, anytime in the entire SW life-cycle. introduced?

Support for various system integration strategies Preliminary software releases are important to allow early system integration. ~~and~~ Software development may be managed with different strategies. It is necessary to respect these strategies and help with the final integration of increments or elements. Satz?

Simplification and harmonization of FDIR Simplification and coordination of the Fault Detection, Isolation and Recovery (FDIR) needs to be handled at both the system and the software level. System engineers and software implementers need to justify the definition of the FDIR strategy at the system level and the software level respectively. A set of functionalities and design patterns need to be provided at the software level that cater to necessary mechanisms for the software realization of the/a FDIR strategy. s. comment above!

Optimize flight maintenance The Flight maintenance may ~~be~~ require~~d~~ to change the OBSW and provision of the required operations and coordination of the strategy to perform it will decrease the time and cost of maintenance. It is better if parts of the software could be updated without having to reboot the CDMU. ??! Satz? Satz

Industrial policy support The development process should enable multi-team software development. It is necessary to incorporate certain flexibility in the allocation of the software elements to industry, according to certain criterion such as prime/non-prime, or geographical return. Multi-team software development is essential to subcontract to non-primes while be in charge of the integration and apply the geographical return policy.

??

Role of software suppliers As discussed before, the new approach must increase the competence of the supplier and foster competition amongst the suppliers: Different suppliers may develop the same component and compete on quality, extension features, performance, cost and schedule. The suppliers will also profit from this approach as they do not have to adapt the software to specific development policies if each single prime.

Dissemination activities System engineers can be exposed to core principle s of the process and if they derive specifications for the system out of the domain of reuse, the costs will certainly increase.

Future needs The trend of increasing complexity of the OBSW gives rise to several needs and these needs need to be subjected to evaluation and their impact on the software reference architecture needs to be monitored. Some of the examples of the future needs include integration of functions of different criticality and security levels, use of Time and Space Partitioning (TSP), support to the multi-core processors, contextual verification of safety properties.

Why is it necessary to read or understand all of this? What is its relevance for the thesis? Where in the later thesis is this needed? In the end a brief summary of the key points is missing!

2.2.4. High level requirements

The user needs ~~were~~ are translated into a set of high-level requirements: Ref. ?!

Software reuse The architecture shall be designed in such a way that the reuse of the functional aspects should be independent of the reuse of the non-functional aspects, reuse of the the unit, integration and validation tests by providing a pre-qualification data package supported by a SW Reuse File in the sense of ECSS software standards. Traced to user needs:

- Shorter software development time
- Reduce recurring costs

Separation of concerns Separation of concerns is one of the cornerstone principles of what? and it deals wit separating different aspects of the software design, in particular the functional and non-functional concerns. Separation of concerns helps to treat ~~reuse~~ functional concerns independently from non-functional concerns and hence increasing the software reuse. Traced to user needs:

- Quality of the product
- Reduce Verification and Validation effort
- Role of software suppliers

??

Reuse of V&V tests The chosen architectural approach should also promote the reuse of Verification and Validation tests that were performed on the software and not just the software itself. The aim is to maximize the reuse of the tests written for the functional part of the component software. Traced to user needs:

- Shorter software development time
- Reduce Verification and Validation effort

?

HW/SW Indep.

HW/SW Independence It enables development of the software independent from ~~the~~ hardware features. Its is necessary to separate parts of the software that interact directly with the hardware into separate modules and make them accessible through defined interfaces. In this way, as long as the interface does not change, the software is isolated from the changes in the hardware-dependent layer. Traced to user-needs:

- Quality of the product
- Mitigate the impact of later requirements definition or change
- Support for various system integration changes

?

Component based approach The whole software is designed as a composition of components that are reusable in nature. The architecture shall respect preservation of properties of individual building blocks once integrated into the architecture and it should be possible to calculate the system's property as a function of components' individual properties. The former is called composability and the latter is called compositionality. More information about this can be found in this section Section 2.3.1. Also, an entire chapter Chapter 3 is dedicated for this topic. This requirement is traced to user needs:

- Shorter software development time
- Reduce recurring costs
- Increase cost-efficiency
- Support for various system-integration changes
- Product policy
- Role of software suppliers

Sofwtare observability The software architecture should provided means to observe the software specific parts and extract current and past status of the software using the services specified by its operational scenarios. This prevents the need for post launch updates or patches of the software in case of failure analysis needs. Traced to user needs:

- Quality of the product
- Reduce Verification and Validation effort
- Simplification and harmonization of FDIR
- Optimize flight maintenance

Software analysability he design process and methodology used for the reference architecture shall support the verification at design time of functional and non-functional properties. Traced to user needs:

- Quality of the product
- Reduce Verification and Validation effort

Property preservation The non-functional properties become the constraints on the system as they specify the "frame" in which the system is expected to behave and be consistent with what was predicted during the analysis. These properties have to be preserved or enforced so that these properties are not only used for the analysis of the software model, but also find their way through to the final system at run-time. Adequate mechanisms should be provided to handle the enforcement of properties and reactions to violation of the properties. Traced to user needs:

- Quality of the product
- Reduce Verification and Validation effort

Integration of software building blocks The architecture should allow the combination of coherent building blocks

- Shorter software development time
- Mitigate the impact of late requirement definition or change
- Support for various system integration strategies
- Product policy
- Role of software suppliers

Support for variability factors In order to reduce the complexity of the architecture, the potential variation of the architecture induced by the variation of the domain must be isolated in some places such as reuse is improved and need for modification is decreased. Traced to user needs:

- Increase cost-efficiency

Late incorporation of modification in the software The architecture should be immune to modification of the software late in the software life cycle. System integration always finds some system problems and it is the responsibility of the software to contain these problems and implement new requirements. The architecture to which the software is conformal to, should be able to handle these late modifications in the software.Traced to user needs:

- Mitigate the impact of late requirement definition or change

Provision of mechanisms for FDIR The aircraft dependability should be handled by the architecture and in particular the Fault Detection, Isolation and Recovery. Traced to user needs:

- Simplification and harmonization of FDIR

Sofwtare update at run-time The reference architecture should allow update to single software components as well as their bindings without having to reboot the entire on-board computer as it is a risk for the system and reduces the mission availability/uptime. Traced to user needs:

- Optimize flight maintenance

2.3. The Software Architectural Concept

COrDeT (Component Oriented Development Techniques) aimed at investigating various techniques in fields such as software product lines engineering, model driven engineering and component orientation. The study came up with the concept of software reference architecture which is to be made up of:

Component Model A component model is the basis for designing the software as a composition of individually verifiable and reusable software units.

Computational Model A computational model is used to relate to the design entities of the component model, their non-functional needs for concurrency, time and space, to framework consisting of analysis techniques, in general, to a set of schedulability analysis equations, which help to judge using formally, whether the description of the architecture is statically analyzable.

A Programming Model A programming model is used to ensure that the implementation of the design entities obey the semantics and the assumptions of the analysis and the attributes used as input to it.

A conforming Exection Platform An execution platform helps to preserve at run-time, the properties asserted by the static analysis, and its able to react to possible violations of them.

2.3.1. Component Model

The software architectural concept is based on component based software engineering (CBSE) and the approach defines a component model that features three software entities: The Component (this is a design entity), the container and the connector (these two entities are used in implementation only and they do not appear in the design space). This approach allows creation of the OBSW as a connection of interconnected components. The execution platform defined in the software architecture provides the services to the components, container and the connectors (refer section Section 2.3.1 for more information). Then finally, all the software is deployed on the physical architecture namely the computational units, equipments, and the network interconnections between them.

Founding principles of choice

This section describes the founding principles of choice of the component model:

Correctness by construction E.W. Dijkstra in his ACM Turing lecture in 1972 suggested that the program construction should be done after a valid proof of correctness of construction has been developed. Two decades later, a software development approach called Correctness by Construction (C-by-C) was proposed which advocated the detection and removal of errors at early stages, which leads to safer, cheaper and more reliable software. The Correctness by Construction practice follows:

- To give a solid reasoning on their correctness,it is necessary to use formal and precise tools and notations for the development and verification of any product item, may it be document or code
- Defining things only once so as to avoid contradictions and repetitions
- Designing the software that is easy to verify eg. by using safer language subsets or using appropriate coding styles and software design patterns.

In OSRA and the component model developed along with it, the Correctness by Construction principle is applied to CBSE approach based on Model-driven Engineering (MDE) approach wherein: