 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N°332830 AND FRO M SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

CRYSTAL Space Toolset Specification
D205.020

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	CRYSTAL Space Toolset Specification
Deliverable No.	D205.020
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-04-30
Contact	Ricardo Moreno Ruano
Organization	TAS-E
Phone	+34 918 07 79 00
E-Mail	Ricardo.MorenoRuano@external.thalesaleniaspace.com

AUTHORS TABLE

Name	Company	E-Mail
Ricardo Moreno Ruano	TAS-E	Ricardo.MorenoRuano@external.thalesaleniaspace.com
Rubén de Juan	ITI	rjuan@iti.es
Ismael Ripoll	ITI	iripoll@iti.es
Elena Alaña	GMV	ealana@gmv.es
Carlos Zubieta	Orbital Aerospace	carlos.zubieta@orbital-aerospace.com
Susana Pérez	Tecnalia	susana.perezsanchez@tecnalia.com
Asier Alonso	Tecnalia	asier.alonso@tecnalia.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	2014/03/25	Creation of document	All
0.1	2014/04/09	Complete TAS-E content	All
0.2	2014/04/16	Added other partners content	15-30
0.3	2014/04/21	Minor changes and format	All
0.4	2014/04/24	Volvo external review comments included	All
1.0	2014/04/28	Final version including AIT external review	All

CONTENT

Content of Tables

Content of Figures

1 Introduction

1.1 Role of deliverable

This document is a specification of the tools required to configure the CRYSTAL Space Toolset as well as a first draft of the application procedure of the CRYSTAL tools to the Space Environment, including design rules, guidelines for the usage of tools and Best Practices. This document is produced after a formal review of the previous requirements and the inclusion of cross domain recommendations and results through internal and external review processes.

1.2 Relationship to other CRYSTAL documents

This document has the following relationships to other CRYSTAL deliverables.

- Extend initial requirements and engineering methods described in D205.010 as well as a more detailed use-case description, including actors and drawbacks of current development process in space domain.
- Establish the technology baseline with respect to the use-case, and the expected progress beyond (existing functionalities vs. functionalities that are expected to be developed in CRYSTAL).
- Provide input to WP601 (IOS Development) required to derive specific IOS-related requirements.
- Provide input to WP602 (Platform Builder) required to derive adequate meta-models.

1.3 Structure of this document

Chapter 2 makes a review of the use case presented in [D205.010], summarizing its rationale, context as well as the applicable standards that need to be followed. It emphasizes those points that needed clarification in the previous deliverable.

Chapter 3 presents the engineering methods that will be implemented during this use case and a potential alignment with respect to those methods of the Public Aerospace Use Case.

Chapter 4 explains in details the bricks that will be used in the use case, including how they will be operated by the associated actor and their interactions with IOS, if any.

2 Use case summary

2.1 Context

In space as well as in the consumer market, as the shrinking of the electronic components enables it, the trend is towards more dense, integrated and reconfigurable systems. Potential reconfiguration of the system (either by upload of SW components or new VHDL design) entails a new scenario where requirements may change during any moment of the project lifetime, covering from design phase to even on-flight operation and maintenance.

To assure the quality of the on-board SW, ESA has developed standards (ECSS series, further explained in section 2.3) defining processes for software engineering (ECSS-E-ST-40C) and software quality assurance (ECSS-Q-ST-80C); these processes are based on a series of customer-client meetings where abundant and exhaustive documentation related to design, analysis and test of the SW product are reviewed. In addition to this, critical SW has to undergo an Independent Software Verification and Validation process (ISVV) by a third party company, thus increasing certification efforts.

European Cooperation for Space Standardization (ECSS) is both a process-based and product-based framework and it allow for a user to choose an own life-cycle and development approach, with appropriate methods and tools. This use case will serve to assess CRYSTAL bricks and technologies (RTPs and IOS) as per the ECSS standards applicability in order to accelerate the development and certification processes of reconfigurable space-qualified systems under its associated standards, thus reducing time and costs efforts. The application to be implemented for the Space domain is the Low Level Software for an Avionics Control Unit, which application software could include autonomous navigation features based on GPS, inertial and/or image acquisition inputs as well as FPGA on-flight reconfiguration control.

2.2 Avionics control unit

Comprising computers, data bus, sensors and actuators and on-board software and algorithms, the avionics subsystem contributes by a huge amount to a given mission's functionality. But it is complex and expensive - corresponding to around 60% of the overall development and verification costs of a typical satellite platform.

A modern Avionics Control Unit includes functions such as:

- DC/DC Power conversion and regulation
- Ground Telecommand Decoding
- Packet Telemetry Formatting
- On Board time management
- Autonomous Reconfiguration
- Local Mass Memory function
- Housekeeping telemetry

- Interfacing with other Avionics subsystems

2.2.1 HW platform

TAS-E has manufactured a dual-FPGA board, which image can be observed in Figure 2-1. This dual-FPGA architecture allows evaluating processing boards for Control Units. The development of the Low Level SW of the Control Unit processor is the main driver of the Aerospace Demonstrator.

The main processor embedded in one of the FPGAs off-loads processing requirements to a second FPGA device that can embed another processor, a DSP or just implement some hardware (VHDL) algorithms.

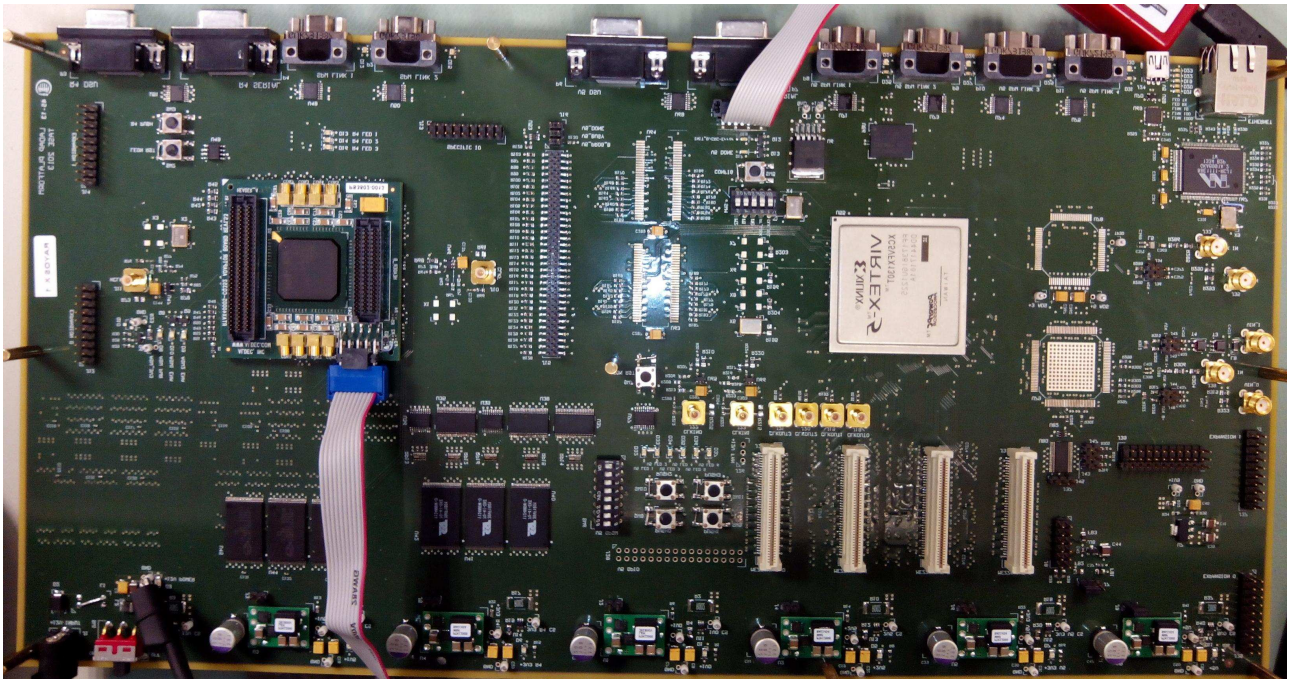

Figure 2-1: Dual-FPGA board

2.2.2 Low Level Software

The Low Level Software supports HW platform initialization and a minimum core of data handling functionality in order to enable diagnostic and load of Application Software (ASW). Although not mandatory, it is traditionally (and is recommended to be) provided together with the HW platform and therefore developed by the HW manufacturer.

It is further divided in Boot, Drivers and Test SW.

1. Boot SW: It is the responsibility of the Boot SW to initialize the board, perform built-in tests, provide health status and launch Application Software from EEPROM memory area to RAM. It is highly critical SW, stored in PROM/EEPROM and not modifiable in flight.
2. Drivers: The drivers' library provides an abstraction layer between HW and other SW components. They are linked as a library to Low Level and Application SW.
3. Test SW: The purpose of this Test SW is to validate the Hardware Dependent Software (HDSW), check correct boot process, check correct communication with the RTOS and

characterize the final Applicative SW behaviour and CPU load when this is not available. Test Software is not included traditionally as flight SW.

Like any other space domain technology, SW is also regulated and must be compliant with its associated standards within the ECSS series explained in the following sub-chapter.

2.3 ECSS series

The set of standards applicable to space domain in Europe are defined by the European Cooperation for Space Standardization (ECSS): a cooperative effort of the European Space Agency, national space agencies and European industry associations for the development of a coherent, single set of user-friendly standards for use in all European space activities.

The main software standard is ECSS-E-ST-40, a part of the ECSS engineering branch (E). It covers all aspects of space software engineering, from requirements definition to retirement. It defines the scope of the space software engineering processes, including details of the verification and validation processes, and their interfaces with management and product assurance, which are addressed in the management (M) and product assurance (Q) branches of the ECSS system.

ECSS-E-ST-40 refers the ECSS-Q-ST-80 for the Software Product Assurance requirements related to the development and maintenance of software for Space Systems. Both apply to any software project procured under ESA contract. The ECSS-E-ST-40 provides a process model for the SW development activities, without prescribing a particular software life cycle.

ECSS is both a process-based and product- based framework, in fact, ECSS is based on “processes”, and allow the user to choose an own life-cycle and development approach, with appropriate methods and tools. This use case will serve to assess CRYSTAL methods and tools as per the ECSS standards applicability in order to accelerate the development and certification processes of reconfigurable space-qualified systems, thus reducing time and costs efforts.

Software dependability and safety are an essential part of ESA programmes, including regular control meeting, technical reviews and documentation that form part of the Product Assurance File. The different software safety and dependability assessment activities are represented in the figure below (Figure 2-2), in relation to the software development, verification and validation activities defined in ECSS-E-ST-40.

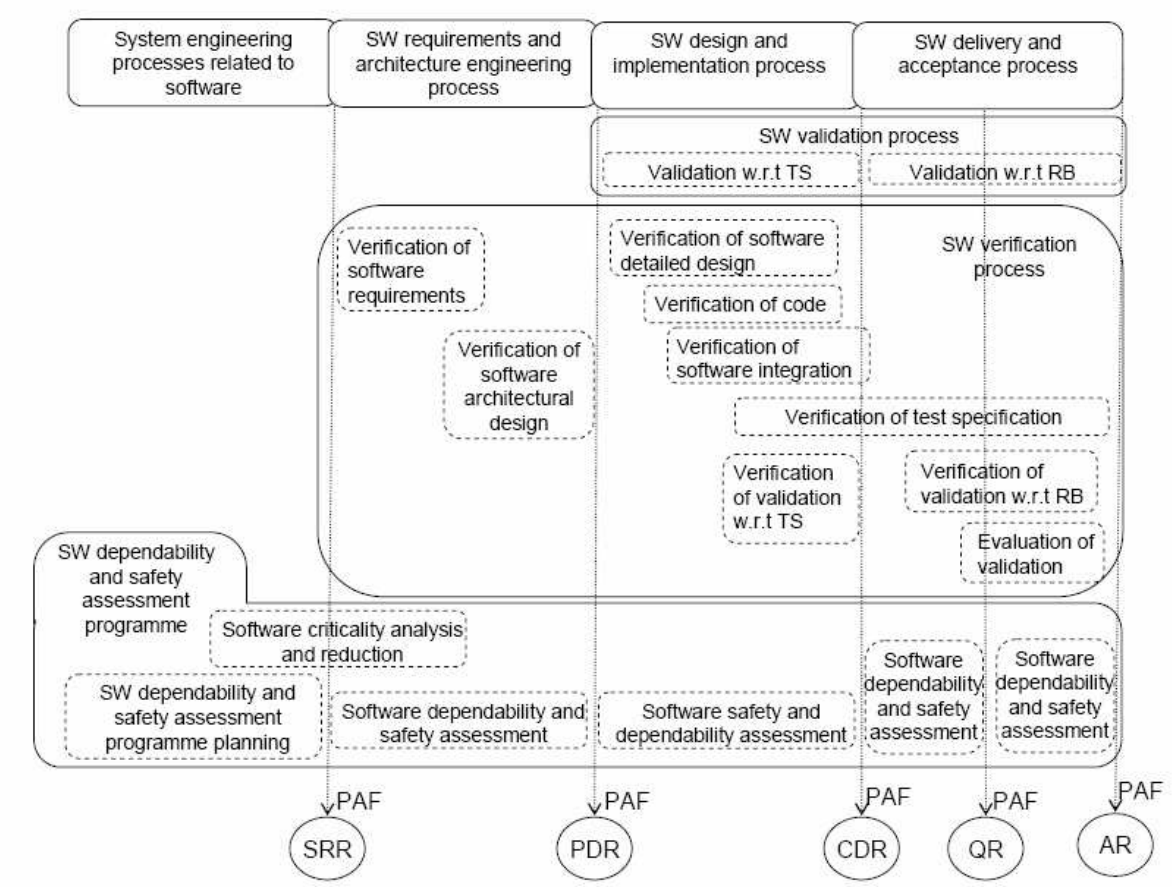

Figure 2-2: Software RAMS activities (ECSS-Q-HB-80)

2.4 Challenges

Besides the classic interactions between the different phases of the product life cycle, some specific requirements for space domain not present in other sectors are: the complexity of the system, the scarce computational resources available in the spacecraft and the radiation doses that on-board electronics receive during each mission lifetime. These facts push the requirements and the selection of the engineering tools, which relationships and artefacts should be perfectly traceable at each point of the development life cycle in order to be compliant with the highly demanding ECSS standards for Product Quality Assurance. These constraints are especially emphasized in the case that new technologies not yet proven in use come into play, as in the case of on-flight FPGA reconfiguration. The following chapter will present the activities to be exercised and improved within the frame of the CRYSTAL project with a set of selected bricks, powering the Safety and Dependability related aspects of a Low Level SW for a HW platform with on-flight FPGA reconfiguration capabilities.

3 Engineering methods

In the previous deliverable [D205.010] we performed a detailed description of the different engineering activities that need to be performed in the frame of a project to be ECSS compliant. Among them, some activities have been selected to be exercised and improved in the frame of the CRYSTAL project with a set of Bricks.

Engineering method	Description	CRYSTAL Associated Brick	Rationale
TS Requirements Analysis	Technical requirements must be specified in a complete, correct, consistent, precise and unambiguous mode.	Partially by AUGE - B2.51	Traditionally performed manually on Word/Excel
Architectural & Detailed Design	To create the architectural and detailed design of the SW.	AFTS-DM – B2.54 Scheduling Requirement Analysis – B2.55	No experience with SoC <-> FPGA reconfigurable systems
Schedulability analysis	To perform the schedulability report results (e.g., WCET).	Scheduling Requirement Analysis – B2.55	Traditionally performed manually with Excel
Validation testing wrt TS	This validation Testing aims to demonstrate that the implementation matches the TS requirements.	AUGE – B2.51	Currently no automatic relationship between Requirements Documentation and Test Description
RAMS analysis.	To assess conformity with respect to a Dependability and Safety analysis.	Safety Analysis for Aerospace – B2.53	No experience in SW RAMS analysis by TAS-E

Table 3-1: Selected Engineering methods

3.1 Stakeholders / Actors

Client: the client role represents an external client defining the product requirements, interface requirements and applicable standards in the form of URD/IRD, or it can also be a Technical Responsible within the company which has already performed a derivation from URD/IRD into different Technical Specification (TS) requirements.

Design Engineer: interpret technical requirements and represent them in a design solution, such as choice of hardware platform, system architecture, application constraints, or into more detailed components, depending on the stage of development.

Test Engineer: validates technical requirements through the definition of test cases and scripts.

RAMS Engineer: analyse requirements, design, tests and results with respect to the applicable ECSS standards of Safety and Dependability all along the lifecycle.

ISVV Engineer: similar to RAMS Engineer but belonging to an external (and independent, thus different to Client and Provider) company.

3.2 Alignment with respect to the Public Aerospace Use Case

An alignment with the Engineering Methods available in the Public Aerospace Use Case has shown up the scenario options presented in the following sections, the final selection of bricks will depend on the evolution of the bricks involved in each case.

3.2.1 General process

The whole process cover the introduction of the requirements by the Client (being internal or external), the Design Engineer having access to these requirements and producing Architecture Design and Schedulability Analysis files of the system. An automatic tool will produce test scripts (substituting Test Engineer role) according to the requirements accessed through IOS. Finally the RAMS Engineer will have access to all the artefacts produced in the several stages and perform an assessment of them as per ECSS Safety related activities and standards.

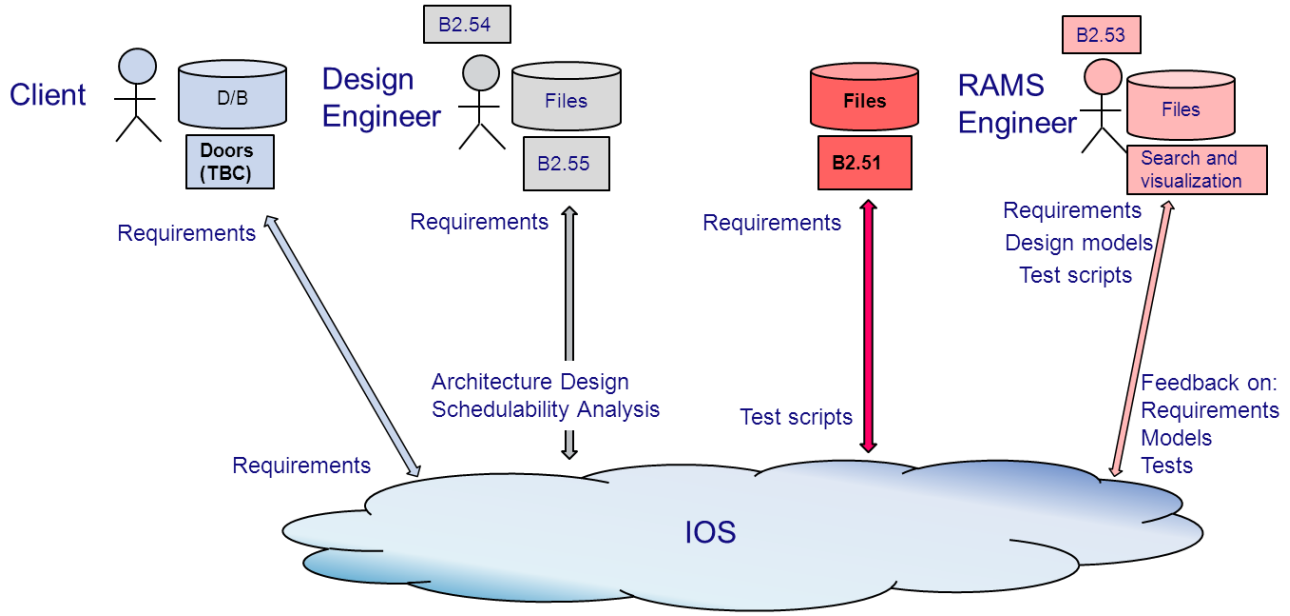

Figure 3-1: Overall general process

3.2.2 Search Data

A subset of the previously general process can be focused in the gathering of data and specific artefacts that the RAMS/ISVV Engineer requires.

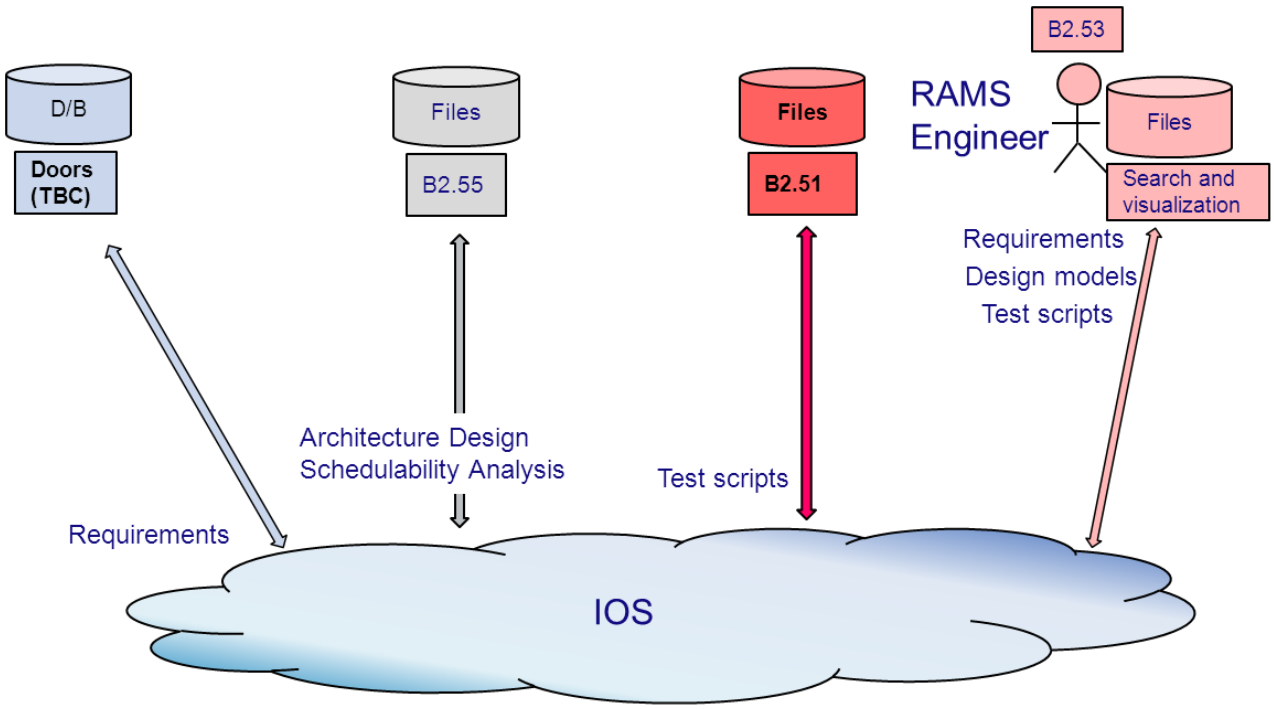

Figure 3-2: Search data

3.2.3 Verify Design and Test against Requirements / Verify Requirements

In this subset the RAMS Engineer collects data and provides feedback on the artefacts he/she considers necessary.

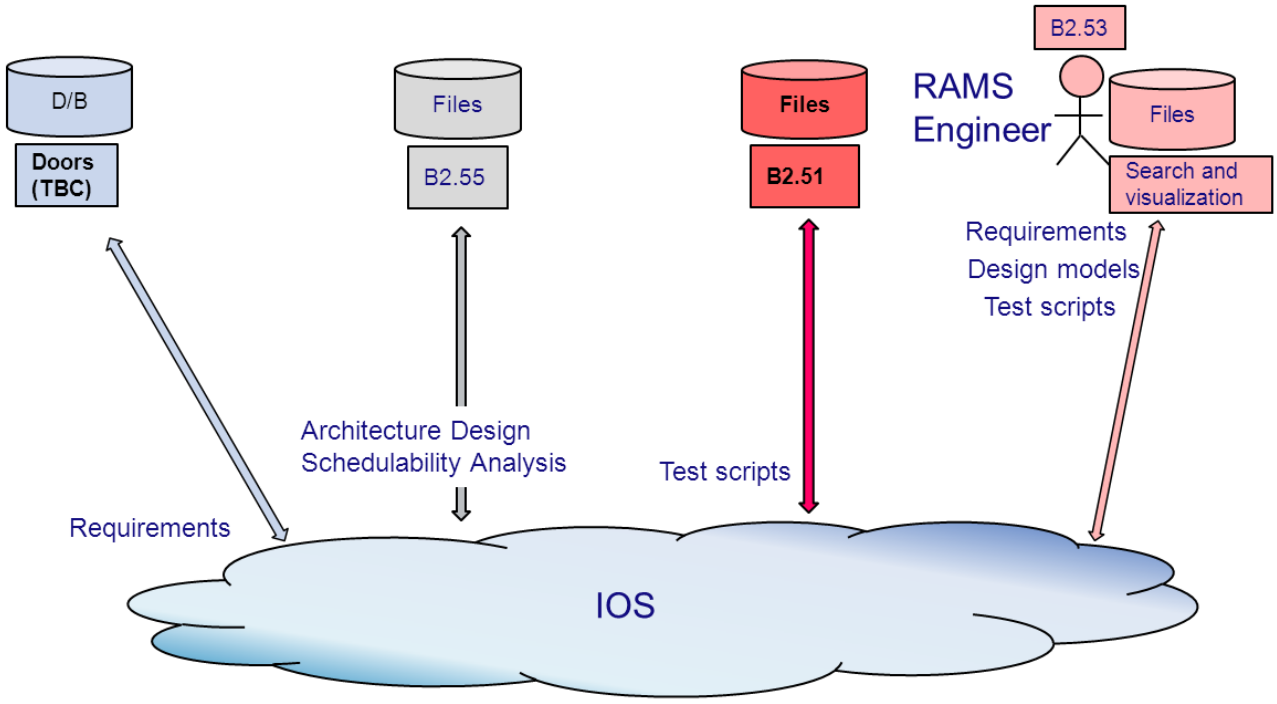

Figure 3-3: Verify requirements step 1

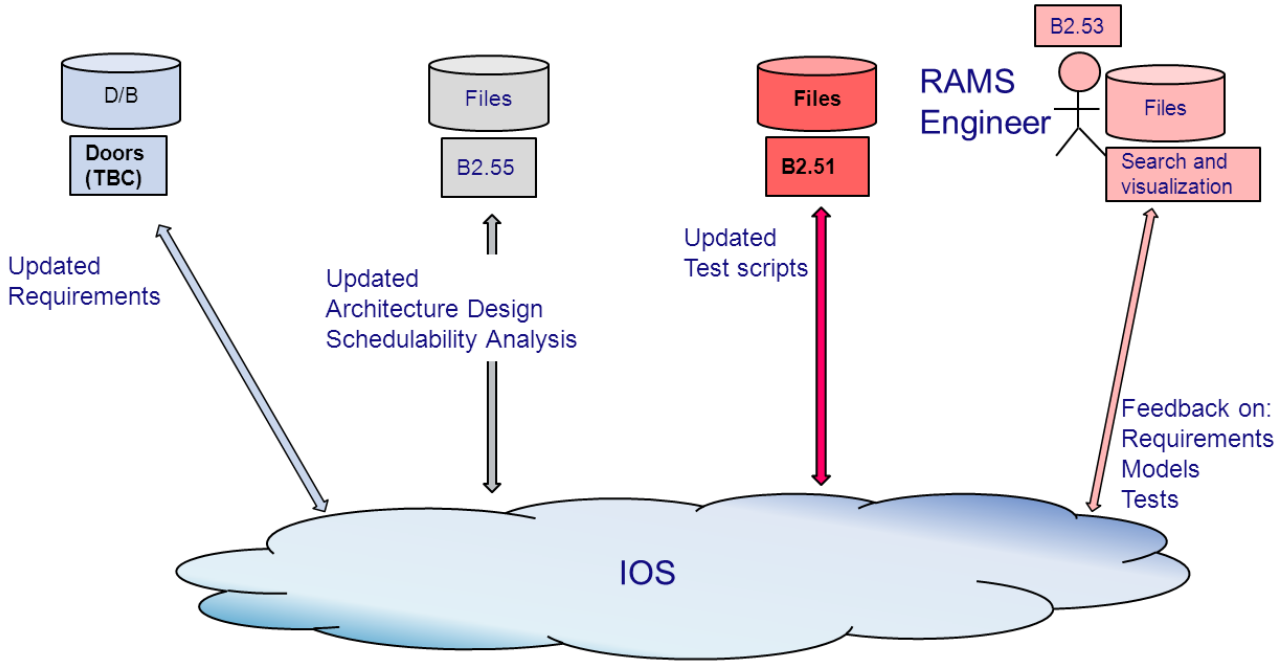

Figure 3-4: Verify requirements step 2

4 Bricks

4.1 AUGE - B2.51

4.1.1 Brick Rationale

Space critical software development normally requires an Independent Software Verification and Validation process (ISVV). In this stage, high-level requirements modules must be traced to related low-level modules, and from these, an additional module refined with convenient test data and syntax is produced and exported to a third-party format (i.e. Microsoft Excel). This set of data is somehow parsed and imported into an Automatic Testing Tool (i.e. TestStand) in order to proceed with the Verification & Validation stage. Thus, the whole process involves heavy data processing which provides no added-value and increases costs and time significantly.

AUGE is a standalone software application that will reduce such costs in Verification & Validation campaigns by achieving automatic test generation from requirements. The tool shall be an independent entity totally driven by IOS-compatible data interfaces and formats, enabling the integration of any kind of Requirements Management Systems and Automatic Testing Tools (provided that these bricks are IOS-enhanced, via plugins, adapters or internal modifications).

4.1.2 Interfaces and Data flow

The following block diagram shows a possible AUGE integration into a Software Development ISVV stage:

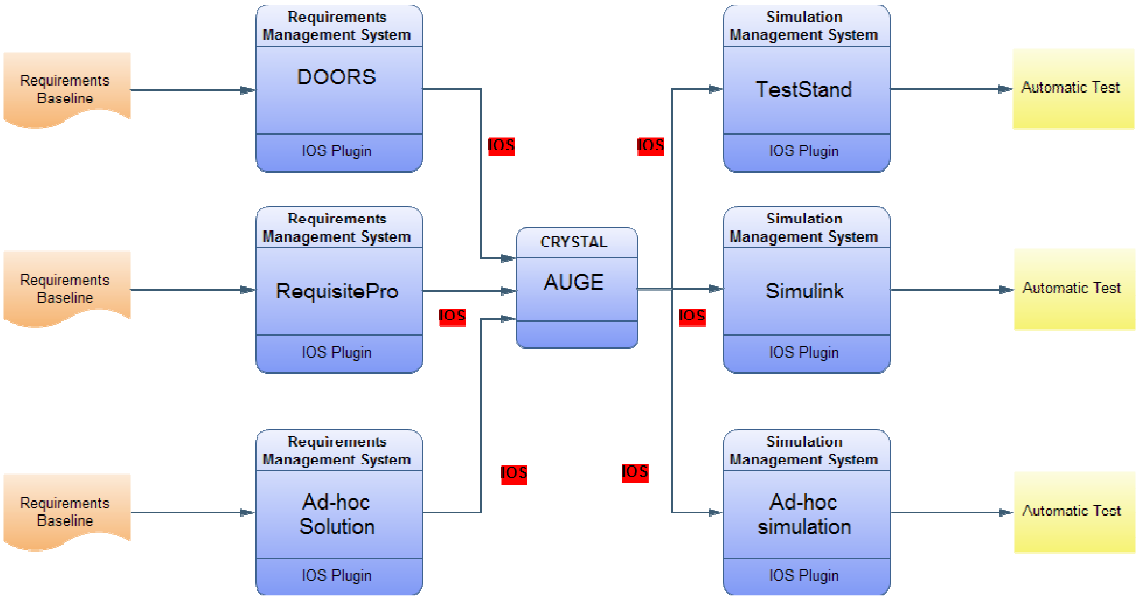

Figure 4.1: AUGE brick data flow

A set of three independent Requirements Management Services, possibly managed by different parties using diverse technologies (i.e. IBM DOORS, RequisitePro or even ad-hoc requirements management software solutions) are deployed to define Requirements Baseline of some (undetermined) Software modules or sub-modules. Provided that all of these Requirement

Management Services implement specific IOS interfaces (defined in future activities throughout the CRYSTAL framework), AUGE brick shall be able to successfully perform a “Browse Requirement Baseline” request and eventually “Retrieve Requirements metadata”. This way, AUGE shall enable ISVV engineers to obtain Requirement Baselines from several remote parties, with independence of the technology or product used to manage the source data.

After successfully retrieving a set of Requirements data (a full Baseline, a subset of Requirements or maybe a Change Proposal), AUGE will proceed to parse the Requirements metadata in order to assess the maturity and validity of each requirement so as to generate an automatic test. This step is the most crucial and sensitive process in AUGE operation, due to the fact that specific Requirement categorization and syntax must be agreed beforehand between parties, not to mention the heavy dependence on specific requirements management standards and procedures of the target engineering use case. Orbital Aerospace was able to provide related know-how in Aeronautics Software development, using the DO-178B/C framework as a starting point for requirements categorization and the specific syntax agreed for parsing. Future compliance with the ESA standard family ECSS-E-40 shall be implemented throughout this Work Package activity.

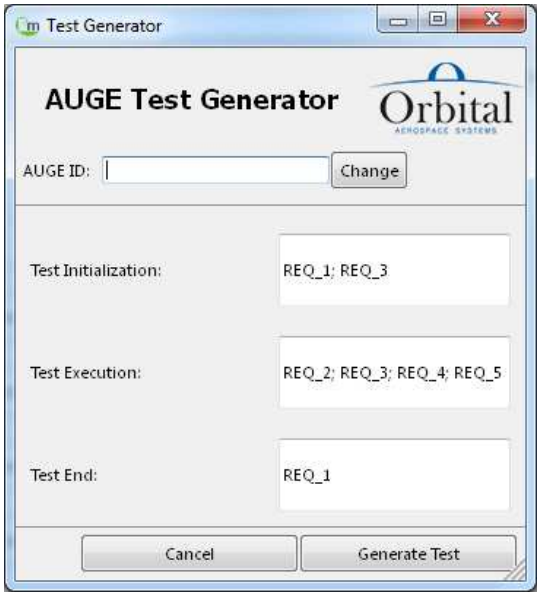

Figure 4.2: AUGE test generator preview

Once a set of requirement data is parsed and validated, AUGE will be able to automatically generate a set of tests which will provide sufficient coverage to verify and validate the requirements. The format of the output test shall be defined in IOS framework as well, instead of focusing on specific testing tools formatting. This way all kind of Testing Framework Tools (TestStand, Simulink, SEAS, etc.) could be used (even remotely).

The whole data flow would achieve a significant reduction in ISVV costs, as complete technology independence would eliminate the necessity of middleware and import/export activities which are often very time-consuming. Additionally, the client-server remote nature of IOS framework (based on WWW and HTTP technologies) would permit remote cooperation between parties with no additional network artifacts required. The bandwidth increase required should be negligible, due to the nature of the information exchanged (standard HTTP queries, plain text data). For Security reasons, all communications shall be made using SSL/TLS protocol scheme.

4.1.3 IOS Interfaces

As previously stated, AUGE shall be designed to totally integrate with IOS: All brick inputs and outputs are IOS-related entities so no linkage to specific commercial tools will be required. AUGE shall contribute to IOS definition with any specific requirements necessary to fulfil the aforementioned tasks.

The following IOS workgroups have been identified to cover AUGE integration requirements:

- Formal Requirements Management
- Change Management
- Documentation Generation

4.1.4 Installation and Usage

AUGE shall be a GNU/Linux native application, distributed as a standard Linux package such as .rpm or .deb file. The package will include all software dependencies (if any) to ease installation process to final user.

In order to use AUGE, a predefined configuration file shall be set by user with all relevant IOS settings such as:

- Requirements Management Server(s) URI(s).
- User credentials (password, profile).

The user shall be able to select a predefined server and browse permitted Requirements modules according to his profile. Once a requirement is selected, the user shall either:

- Preview the Requirement. A new dialog window shall display some requirement data, as defined in OSLC “Linking Data via HTML User Interface” specification.

- Generate Test. The Requirement data shall be processed by the application. If the data syntax is correct the requirement will be considered mature and an output text file shall be generated containing an automatic test defined in IOS-defined syntax. Otherwise, the requirement shall be considered not mature and an error message shall be displayed.

4.2 Safety Analysis for Aerospace - B2.53

Brick B2.53 evaluates the industrial applicability of the safety-analysis framework in the scope of space systems.

The functionality of space systems is increasing more and more in size and complexity. This creates the need for the adoption of appropriate techniques, methods and procedures for the identification and assessment of RAMS (Reliability, Availability, Maintainability and Safety) requirements:

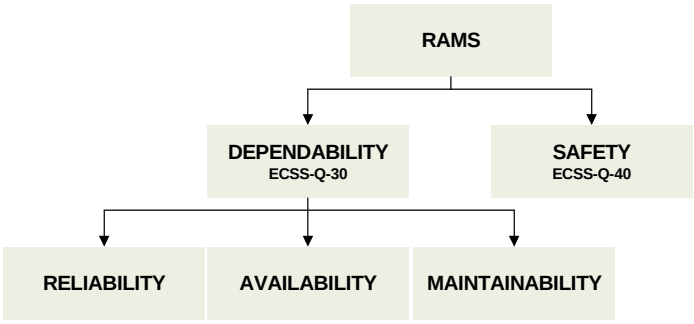

Figure 4-1: RAMS concept

- *Reliability*: Probability of a system to perform without failures for a specific period of time under given conditions. It measures the continuity of service.
- *Availability*: Probability of a system to be properly operating when requested for used. It measures the readiness for usage.
- *Maintainability*: Easiness of repairing the system after a failure or upgrading.
- *Safety*: The ability of the system to operate without catastrophic failures.

Whereas dependability deals with the avoidance of failures, safety focuses on the avoidance of a specific class of failures (those with catastrophic consequences on the users and environment). Dependability can be increased by removing faults that are not concerned with safety. Safety is directly concerned with the consequences of failures, not merely with the existence of failures. Although both activities may overlap in a number of aspects, they are different areas of engineering oriented to different aspects of the software:

- Malfunction aspects for Dependability.
- Harm aspects for Safety.

They can use different or similar techniques. In general, safety has a broader scope than failures of the software, and dependability analyses also those aspects leading to failures but not compromising the safety. The overlapping makes the activities closely related since many of the concepts, techniques, and tools are common.

4.2.1 Dependability and safety process

RAMS requirements need to be defined and subsequently assessed in the system design, implementation and verification and validation process.

The dependability and safety process allows the RAMS Engineer to verify the implementation of RAMS requirements and to mitigate the dependability and safety risks. It is an iterative and continuous process that provides dependability and safety design guides. It has to be conducted along the project lifecycle at every development phase.

RAMS analysis methodology can be applied during the entire software life cycle: from requirements definition to design, implementation, operation and maintenance. It is important to identify RAMS requirements and check that requirements are met to control and reduce risks effects.

Safety is expected to be considered at system level. Software, as part of the system, becomes safety critical when used to control potentially dangerous (parts of the) systems. Instead, dependable software exists in itself when software is available or reliable as specified by the system to which it belongs.

There are different methods and techniques to support this process, such as:

- Failure Modes Effects and Criticality Analysis (FMECA).
- Fault Tree Analysis (FTA).
- Hardware-Software Interaction Analysis (HSIA).
- Hazard Analysis (HA).
- Common Cause Failure Analysis (CCF).

Figure 4-2 illustrates an example of the safety assessment activities linked to each phase of the software V-lifecycle. As it is shown, RAMS analysis runs in parallel with the whole lifecycle. In general, RAMS analysis can run either in parallel or in conjunction with the system development process.

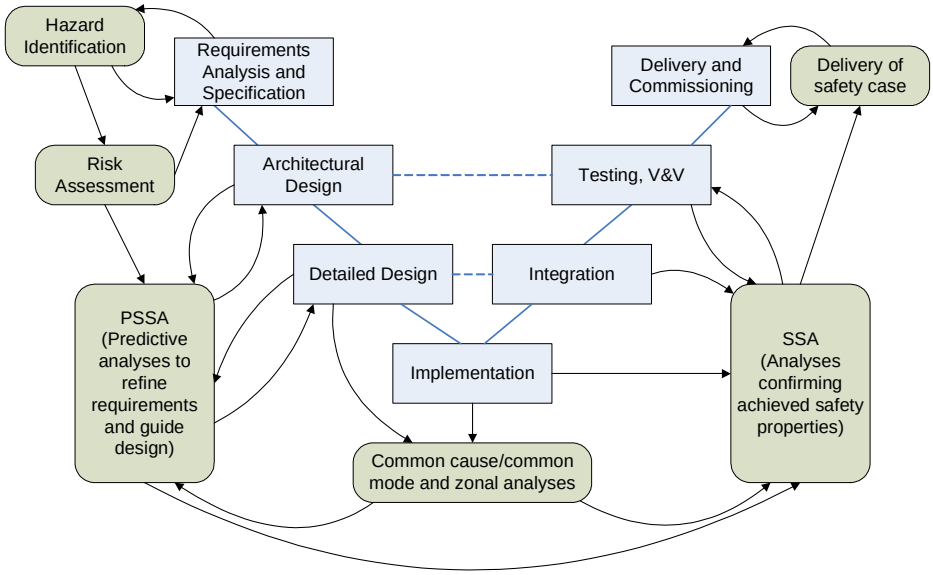

Figure 4-2: V-lifecycle model including the safety activities

To carry out an efficient software dependability and safety analysis two different techniques can be used:

- Inductive Bottom-Up approach: from identified failures, the hazards that could arise from them are assessed (e.g., FMECA technique).
- Deductive Top-Down approach: starts analysing identified hazardous-events with the purpose of finding out the potential causes (e.g., FTA technique).

A combination of both techniques provides a more complete view trying not to overlook possible failures.

The coordination among dependability and safety activities is essential from the very beginning of the project. Analyses are basically applicable to both fields and need to be performed in close synchronization.

The dependability and safety process will be compliant with the following ECSS standards:

- **ECSS-Q-ST-30C** defines the requirements for a dependability assurance programme in space projects. This standard calls for the use of dependability analysis techniques, tailored to match the generic requirements in each project, to address the hardware, software and human functions composing the system.
- **ECSS-Q-ST-40C** defines the safety programme and the technical safety requirements for space projects.
- **ECSS-E-ST-40C** defines the principles and requirements applicable to space software engineering. In its last version (version C), it assets the need of specifying software RAMS requirements based on the System RAMS analysis result.
- **ECSS-Q-ST-80C** presents the software product assurance requirements to be met in a particular space project to provide confidence to the customer and to the suppliers. Namely, ECSS-Q-ST-80C presents:

- Requirements to ensure that the software is developed to perform as expected and safely in the operational environment, meeting the quality objectives agreed for the project.
- Requirements concerning “Software dependability and safety analysis” (subclause 6.2.2). These requirements (through the referred requirements of ECSS-Q-ST-30C and ECSS-Q-ST-40C) refer to the supplier carrying out a software dependability and safety analysis to assign criticality levels to software components, based on the criticality levels of the functions and the identification of safety functions. In addition, subclause 6.2.2 of ECSS-Q-ST-80C mentions that the software dependability and safety analysis is performed at every development milestone. It also expects that the list of software critical components is verified, reviewed and reduced and designed to facilitate dependability and safety analysis and software testing.
- Requirements concerning “Handling of critical software” (subclause 6.2.3), regarding measures and activities to ensure dependability and safety of critical software components, the verification of the use of those measures, what to do regarding dead code and about non-critical code potentially affecting the critical code.

4.2.2 Use case

This brick will identify the suitable methods and techniques to conduct the dependability and safety analysis on the Aerospace Use Case. The process will be in line with the ESA standards (ECSS-Q-ST-30C and ECSS-Q-ST-40C).

Firstly, the dependability and safety process will be tailored according to the criticality level of the use case. Then, based on this criticality, the suitable engineering and product assurance measures (i.e., techniques and methods) will be applied.

The main actor involved in the execution of this brick is the RAMS Engineer. A further analysis can be performed to distinguish between the Safety and Dependability Team. For example, in case of identifying the safety information required to perform the initial safety risk assessment of the identified hazards:

- Safety Team provides the initial hazard analysis.
- Dependability Team provides the failure modes analysis.

All the input artefacts needed to generate the dependability and safety analyses shall be extracted from the IOS. The RAMS Engineer shall access the IOS to extract the requirements, architecture design, schedulability analysis, and test scripts and then perform the subsequent dependability and safety analysis. This process will be done along the whole use case phases (e.g., requirements, design, etc.).

This way, the brick will provide the following results to the use case:

- Dependability and safety analysis.
- Assessment of the artefacts obtained through the IOS (e.g., additional artefacts needed, artefacts which shall provide additional information, etc.).
  - It will provide feedback concerning the adequacy of the outputs of other bricks with respect to the ECSS dependability and safety standards.
  - Assessment of the artefacts along the several stages of the development life-cycle.
- Assessment of the adequacy of the process with the dependability and safety standards.

4.3 AFTS DM - B2.54

4.3.1 Concept

Fault tolerance is intended to preserve the delivery of correct service in the presence of active faults. It is generally implemented by error detection and subsequent system recovery, and possibly by error containment.

- Error detection: Originates an error signal or message within the system.
- Recovery: transforms a system state that contains one or more errors (and possibly faults) into a state that can be activated again without detected errors and faults. Recovery consists of error handling and fault handling.
    - Error handling: eliminates errors from the system state.
    - Fault handling: prevents a fault from being activated again in

A remark may be added about the service provided by the system. Here, we consider that a correct service has a behaviour in accordance with the intended system function from a user point of view. It is defined before execution and is embodied in the system specifications, which determines the goals to be achieved in well-defined situations. A dependable system should thus provide a correct service regarding nominal situations and explicitly-specified adverse situations.

However, an autonomous system is often required to function in an open environment, where all operating conditions can not completely be determined in advance. When faced with unexpected adverse situations a correct service cannot be guaranteed.

The final objective is to create tools and technologies to help systems adapt to their environment without human intervention. Ideally, these systems are able to deal with problem specification changes and respond to unexpected input signals variations, changes in conditions like energy availability, bandwidth adaptation and many others. Among them, fault tolerance could greatly benefit from the evolving hardware approach, which can be considered as an important technology to provide systems with self-healing capabilities.

Reconfiguration is a key technique to provide systems with adaptability, bringing the adaptive hardware challenge nearer.

Dynamic Partial Reconfiguration or Partial Run-Time Reconfiguration, hereafter referred to as Partial Reconfiguration (PR), is a process consisting of swapping parts or modules of a reconfigurable system while the rest of the systems remains running and therefore fully operational. Some portions of the FPGA logic, referred to as Reconfigurable Region (RR), are modified dynamically by downloading a partial bitstream file or Reconfiguration Modules (RM) through the configuration port. During the PR process, the rest of the system or logic, referred to as Static Region (SR), continues running without being affected. The reconfigurable logic is therefore replaced by the content of the partial bitstream.

4.3.2 Requirements

The main requirement for the employment of the AFTS DM engineering method is that it is only planned for Xilinx devices. As shown in Figure 2-1, there are two processing elements in the hardware platform of the use case, an ACTEL RTAX4000 FPGA and a Xilinx Virtex 5. Therefore, in the context of the use case only those applications implemented in the Virtex 5 device can be reconfigured.

A preliminary analysis is needed to choose the application modules which will be assigned to each of the configurable devices. Those which are chosen to be implemented in the Virtex 5 device are affected by the described engineering method.

4.3.3 Impact on the architectural design

In order to allow partial reconfiguration in the Virtex 5 device, a SoPC (System on Chip Partially Configurable) architecture will be employed in the SR. This architecture comprises a processor and required peripherals to implement PR. When the processor detects a malfunction of any of the implemented modules (RM), PR techniques are employed to ensure self-healing of the system; additionally PR can be applied on an external stimulus from the other processor (ACTEL RTAX device). Thus, the correct service of the system is guaranteed.

The following figure shows the graphical depiction of the SoPC architecture, the SR is depicted in blue and the RR is depicted in orange:

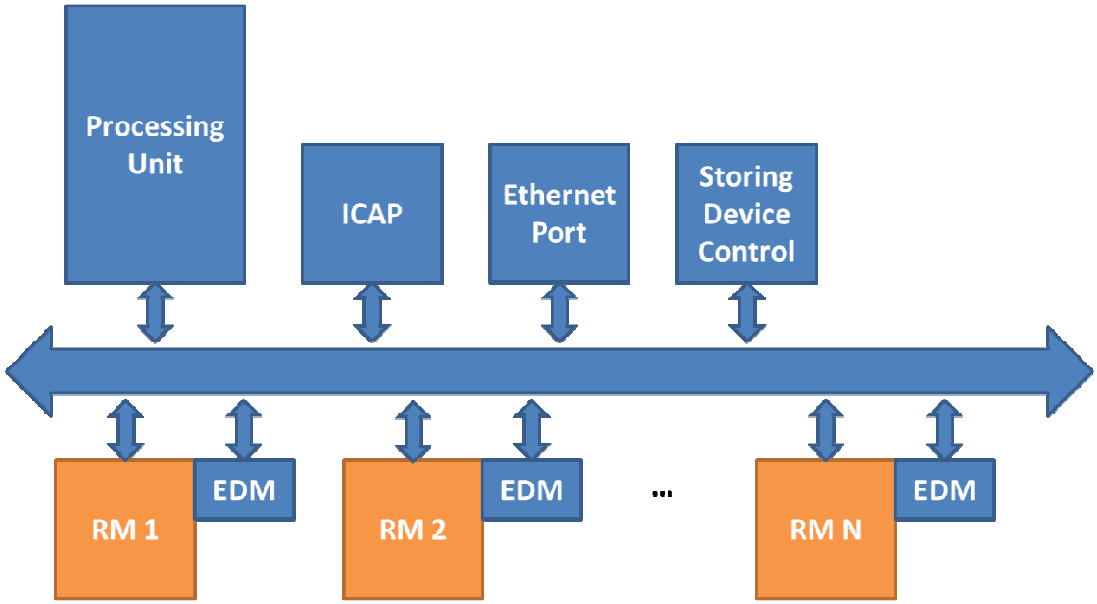

Figure 4-3: Architecture of the SoPC

The SR consists of the following modules:

- **Processing Unit**: The Processing Unit is the Central Processing Unit (CPU) of the SoPC. The software in charge of controlling the PR is executed by the Processing Unit.
- **ICAP**: Partial bitstreams are loaded from the storing device to this IP-Core in order to accomplish the PR.
- **Ethernet Port**: or other alternative communications interface, is used to connect the FPGA with an external processor for debug and command purposes. External stimulus to proceed with the PR will be received through this port.
- **Storing Device Controller**: This IP-Core is a controller for an external storing device such as a flash memory or an external card. The device stores both the partial and global bitstreams required to configure the FPGA.
- **EDM**: Error Detection Module, one for each RR. They are in charge of detecting errors in the different modules and notifying them to the processing unit.

The dynamic section consists of different RRs intended to host the RM. The architecture of this part, including defining the different modules and the interfaces and interactions among them is to be designed by the partners involved in the use case.

4.3.4 Impact on the detailed design

To design a PR system, first of all, the designer must define both the SR and at least one RR. With reference to the latter, the designer defines an RR in terms of both physical size and type of resources required. For each RR, a different set of RM is considered. It must be taken into account that the quantity and type of resources provided by the RR must be sufficient to host each of the selected RMs.

The SDK software ensures that the resources used to construct the RM are completely contained within the selected RR and that no interference with the SR exists. Communication between static logic and reconfigurable logic is accomplished via the so-called Proxy Logic. A Proxy Logic is a single LUT element automatically inserted by the software for each port of an RM (referred to as Partition Pin).

There are different PR styles depending on the way that RMs are implemented onto the RRs.

- Island style: This style allows swapping a set of RMs exclusively in their assigned RR on the FPGA. Although more than one RR may exist, each island is bound to host its individual set of RMs (one at a time), thus not being possible to swap RMs between RRs. This configuration style is the only one supported by FPGA vendor tools such as Xilinx PR Design Flow.
- One-Dimensional Slot style: In Island style, the largest RM to be hosted by an RR defines the RR size. This results in a resource waste when a smaller RM is to be implemented in that RR, because a large RM cannot be replaced by multiple smaller RMs. This effect is referred to as internal fragmentation.. Aiming at improving resource utilization; the RR is divided into a set of adjacent one-dimensional aligned resource slots. Hence, RMs can be implemented using the required number of adjacent slots.
- Two-Dimensional Grid style: Even making resource slots as narrow as possible (1 CLB column wide in case of Xilinx FPGAs), it can still result in a waste of resources. It occurs in particular when dedicated primitives such as RAMs or multipliers are required. A step further consists in dividing the resource slots so that they are organized in a two-dimensional grid. Thus, the internal fragmentation is reduced but in contrast RM placement becomes more complex.

PR system design support is provided via PlanAhead tool provided within the SDK software. All the elements required to build a PR system (SR, RRs and RMs) are managed in Xilinx PlanAhead. Floorplanning, required to define RRs, and Design Rule Checks (DRC), established to guide designers on a successful path on design completion, are all accessed through the Xilinx PlanAhead software environment.

4.4 Scheduling Requirements Analysis - B2.55

Brick B2.55 is a design and analysis tool, Art2kitekt, specially designed for a target sector, the aerospace in this UC. The tool is designed to follow the same ontology used by the aerospace sector; the expressiveness, abstractions and generality will be limited so that there is a direct

binding between the modelled elements and the finally implemented elements. The underlying model is a subset (a restricted pattern) of the AADL standard.

This brick is used in the TAS-E use case for modelling and analysing the proposed system hardware and software. The following sections detail the brick structure and its main features and the envisaged use in the WP 205 use case.

4.4.1 Brick Structure and Features

The brick B2.55 can be seen as a tool suite for Scheduling Requirement Analysis that is composed of the following elements:

- A system architecture model server, based on AADL. This server will provide features for storing, querying and retrieving AADL models to the corresponding client tools. The architecture model server (AMS) will be a server process (daemon) which provides the IOS front end to the AADL specification. It can read the model from a textual or XML AADL file. The first versions will only operate with the complete model. In the future versions it shall allow CRUD operations over parts of the model. This fine grain access granularity will be developed once a more elaborate version of the IOS specification for the architecture model is available.
- The AADL modeller tool. This is the editor component, which will guide the user (engineer) in the model creation by restricting the modelling capabilities, so that it matches the model that can be analysed by the analyser tool. Usually the editor will provide more design flexibility in the software side than in the hardware part. In this way, the modeller will provide specific interfaces, used as a wizard, in order to guarantee a proper modelling of the system. This tool will store and retrieve the models from the AMS. Moreover, this tool will be able to interact through IOS with a Requirement Management Tool for collecting requirements and map them to the model components supporting or contributing to them.
- The Schedule (model) analyser. This component will provide a custom analyser for the use case defined by TAS-E. This tool may be integrated with the main editor if possible (using delegated UIs). The output of the analyser will be:
    - reported to the user
    - written back to the architecture model server to annotate the model,
    - generate the configuration files, if applicable.
    - generate code or the basic execution framework, if applicable.

  This tool may also interact –this feature of the tool is still under discussion- with a Requirement Management Tool through IOS in order to mark which requirements are satisfied by the design and which aren't.

Next picture shows the main elements of B2.55.

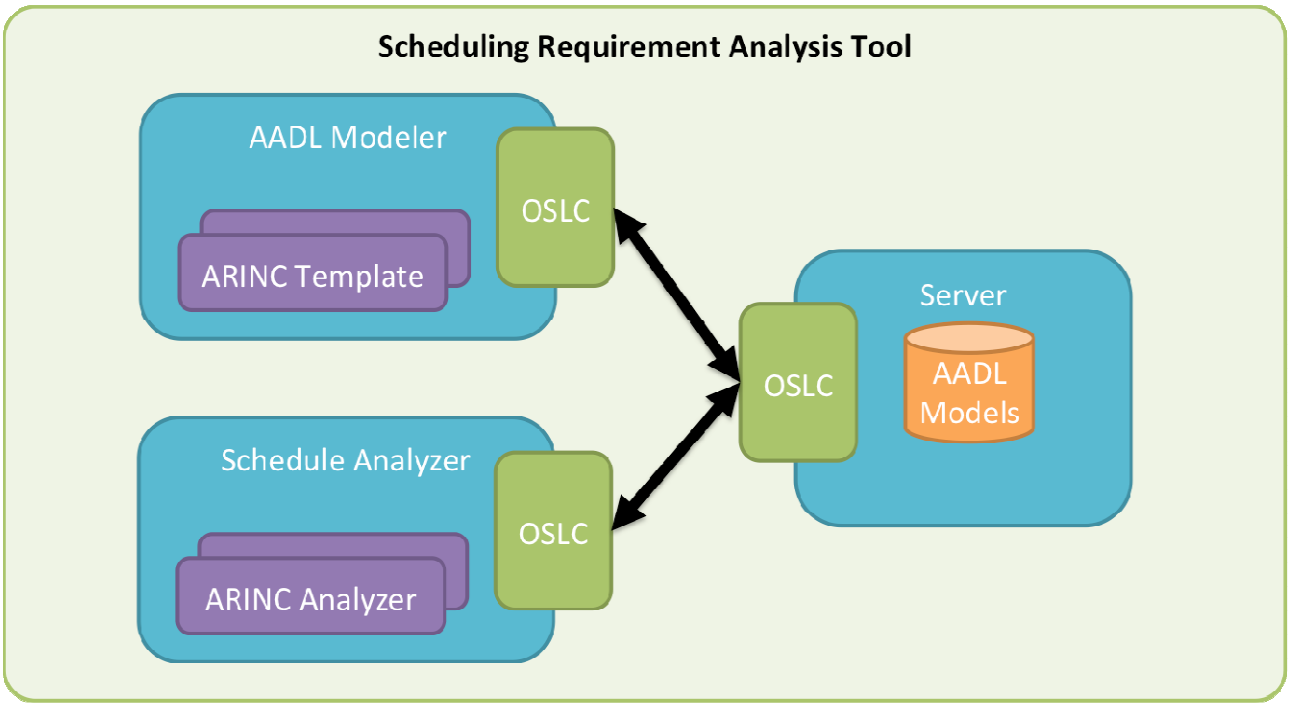

Figure 4-4: B2.55 decomposition

4.4.2 Brick Usage

The B2.55 tool is used for the following purposes in the WP205 use case:

- System modelling and early testing
- Linking model elements to requirements
- Model analysis and product certification

The following sections details each one of those purposes. They are listed in this order because it is the most natural one.

4.4.2.1 System Modelling

Usually once Requirements are defined the engineer will start the process of designing the system to be obtained. In order to do so, the engineer will follow these steps:

1. Open the Modeller Tool.
2. Create a new model or open an existing one (importing it from the AMS).
3. Use the provided wizard for modelling the hardware characteristics of the platform.
4. Use the provided wizard for modelling the software to be run in the hardware defined.
5. Store the obtained model in the AMS.

4.4.2.2 Linking Model Elements to Requirements

Parallel to the modelling or once it is finished the engineer will be able to link the hardware and software components to the requirements they are covering or satisfying.

1. Open the Modeller Tool.
2. Open a model (importing it from the AMS).
3. Retrieve the list of requirements from the Requirement Management Tool using IOS.
4. Select a requirement and map it to the component of the model.
5. Store the annotated model in the AMS.

4.4.2.3 Model Analysis

Once a model has been created (mapped to requirements or not) it can be analysed in the Schedule Analyser tool. The steps to be performed are the following. This behaviour can vary if finally the Schedule Analyser is integrated with the Modeller.

1. Open the Schedule Analyser Tool.
2. Open the model (importing it from the AMS). Or part of the model (for future versions).
3. Start the analysis. Adjust the model parameters and the analysis options to generate a model that matches the requirements.
4. Store the annotated results in the AMS.
5. Check the results.
    a. Depending on the results the design should be refined / improved. If this is not possible maybe requirements should be changed (using delegated UIs from the Requirement Management Tool).
    b. If the result of the analysis is good enough the model can be used as it is for starting the implementation phase.

At the end of all those sections the engineer has obtained a model that has been early validated in regard to the fulfilment of real-time requirements.

4.5 Bricks outside WP205

The following bricks (or brick-elements) have been identified as recommended/required in order to obtain the maximum benefits from CRYSTAL project.

4.5.1 Requirements tool

A requirements management tool, through which requirements are gathered and accessible by the different actors, is necessary in order to set up a complete and representative use case. Contacts have been initiated with IBM in order to assess the use of IBM Rational Doors to cover this gap. In case such tool cannot be included in the demonstrator (i.e: due to licensing issues) a work-around will be implemented.

4.5.2 Search and Visualization Engine

A tool for searching key elements and artefacts through different databases and file formats is required in order to assist the activities of the RAMS Engineer role. Such a tool has not yet been identified among the tools within the project.

5 Terms, Abbreviations and Definitions

AADL	Avionics Architectural Design Language
AFTS-DM	Autonomous Fault Tolerant System - Design Methodology
AMS	Architecture Model Server
ARINC	Aeronautical Radio, Incorporated
AUGE	AUtomatic GEneration test tool
CCF	Common Cause Failure Analysis
CO	Confidential, only for members of the consortium (including the JU).
CLB	Configurable Logic Block
CRYSTAL	CRitical SYSTem Engineering AcceLeration
CRUD	Create, Reade, Update and Delete
DRC	Design Rule Checks
DSP	Digital Signal Processor
ECSS	European Committee for Space Standardization
EDM	Error Detection Module
ESA	European Space Agency
FMECA	Failure Modes Effects and Criticality Analysis
FPGA	Field Programmable Gate Array
FTA	Fault Tree Analysis
HA	Hazard Analysis
HDSW	Hardware Dependent SoftWare
HSIA	Hardware-Software Interaction Analysis
ICAP	Internal Configuration Access Port
IOS	Inter-Operability Specification
IP	Intellectual Property
IRD	Interface Requirement Document
ISVV	Independent Software Verification & Validation
LUT	Look-Up Table
OSLC	Open Services for Lifecycle Collaboration
PR	Partial Reconfiguration
R	Report
RAMS	Reliability, Availability, Maintainability and Safety
RM	Reconfiguration Modules
RR	Reconfigurable Region
RTOS	Real Time Operating System

RTP	Reference Technology Platform
SDK	Software Development Kit
SoC	System-on-Chip
SoPC	System on Chip Partially Configurable
SR	Static Region
TAS-E	Thales Alenia Space - España
TS	Technical Specification
UC	User Case
URD	User Requirements Document
URI	Uniform Resource Identifier
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WCET	Worst Case Execution Time
WP	Work Package

Table 5-1: Terms, Abbreviations and Definitions

6 References

[ESA-E-40, 2009]	ESA; *Space Engineering - Software*; ECSS-E-ST-40C(6March2009)
[ESA-Q-80, 2009]	ESA; *Space Product Assurance - Software Product Assurance*; ECSS-Q-ST-80C(6March2009)
[DOW Crystal, 2013]	Crystal JU; *Annex I – Description of Work*; DOW CRYSTAL (332830) Part_A_B 2013-02-28
[D205.010]	Crystal WP205; *Space Use Case Requirements D205.010*