Due date of deliverable: 31-03-2017  
Start date of project: 01-11-2016  
Duration: 27 months  
Topic: COMPET-4-2016 Building Block a) Space Robot Control Operating System  
Work package: WP2100  
Lead partner for this deliverable: GMV

“This project has received funding from the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement No 730080.”

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
</table>
| PU Public           | X  
| CO-1 Confidential, restricted under conditions set out in Model Grant Agreement. Version providing the PSA will all the information required to perform its assessment. |  
| CO-2 Confidential, restricted under conditions set out in Model Grant Agreement. Version providing the PSA and the other operational grant the information required for the integration of all the building blocks and the continuity of the Strategic Research Cluster |  

Prepared by: ESROCOS team  
Approved by: GMV  
Authorized by: Jorge Ocón

Code: ESROCOS_D2.1  
Version: 1.0  
Date: 31/03/17  
Internal code: GMV 21188/17 V1/17  

© ESROCOS Consortium 2017, all rights reserved
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................................................................. 5  
   1.1. PURPOSE ...................................................................................................................................... 5  
   1.2. SCOPE ....................................................................................................................................... 5  
   1.3. CONTENTS ................................................................................................................................... 6  
2. REFERENCE AND APPLICABLE DOCUMENTS ...................................................................................... 7  
   2.1. APPLICABLE DOCUMENTS ......................................................................................................... 7  
   2.2. REFERENCE DOCUMENTS ........................................................................................................... 7  
3. TERMS DEFINITIONS AND ABBREVIATED TERMS .......................................................................... 8  
   3.1. DEFINITIONS ............................................................................................................................... 8  
   3.2. ACRONYMS ............................................................................................................................... 8  
4. PRODUCT OVERVIEW ........................................................................................................................... 10  
5. USE OF ESROCOS IN SPACE ROBOTICS .............................................................................................. 11  
6. ENVIRONMENT PREPARATION TOOLS ............................................................................................. 13  
7. THIRD-PARTY SW INTEGRATION TOOLS .......................................................................................... 15  
8. ROBOTICS MODELLING TOOLS ......................................................................................................... 17  
9. TASTE MODELLING ............................................................................................................................. 19  
10. BIP MODELLING TOOLS ..................................................................................................................... 20  
11. IMA MODELLING ............................................................................................................................... 21  
12. CODE GENERATION TOOLS ............................................................................................................. 23  
13. ADDITIONAL SW ASSETS ................................................................................................................... 24  
14. ESROCOS OVERALL ARCHITECTURE ............................................................................................... 25
LIST OF TABLES AND FIGURES

Table 2-1: Applicable documents ........................................................................................................... 7
Table 2-2: Reference documents ............................................................................................................... 7
Table 3-1: Definitions ............................................................................................................................... 8
Table 3-2: Acronyms ................................................................................................................................. 8

Figure 1: Activities associated to SW development, described in ECSS [RD. 17] ......................... 11
Figure 2: Public website of ESROCOS .................................................................................................. 14
Figure 3: Example of TASTE Transformations and Code Generation ................................................. 15
Figure 4: Geometric primitives useful to define the position, orientation, pose, linear velocity, angular velocity, and twist of body C with respect to body D (left) and KDL robotic arm modelling (right) ................................................................................................................................. 17
Figure 5: TASTE interface view (top) and deployment view (bottom) .............................................. 19
Figure 6: Partition Schedule in Modular architectures ......................................................................... 21
Figure 7: AIR multi-core example schedule ......................................................................................... 22
Figure 8: ESROCOS high-level architecture ......................................................................................... 25
1. INTRODUCTION

1.1. PURPOSE

The PERASPERA OG1 activity is devoted to the design of a Robot Control Operating System (RCOS) that can provide adequate features and performance with space-grade Reliability, Availability, Maintainability and Safety (RAMS) properties. The goal of the ESROCOS project is to provide an open source framework that can assist in the development of flight software for space robots. By providing an open standard that can be used by research labs and industry, it is expected that the Technology Readiness Level (TRL) can be raised more efficiently, and vendor lock-in through proprietary environments can be reduced. Current state-of-the-art robotic frameworks are already addressing some of these key aspects, but mostly fail to deliver the degree of quality expected in the space environment. In the industrial robotics world, manufacturers of robots realise their RCOS by complementing commercial real-time operating systems, with proprietary libraries implementing the extra functions.

In this document we describe the main characteristics of the Product that will be produced as main outcome of the ESROCOS activity. This document has been produced 2 months after the SRR has been held, and has been done with the information available at this time. Since ESROCOS is a live project, final details about the tools and the framework might have changes in the future. However the information provided is considered enough as to give an accurate overview of its capabilities and characteristics.

1.2. SCOPE

ESROCOS is based on existing SW assets and newly developed assets in an orchestrated manner, used to cover the requirements of the ESROCOS product documented in [AD.5]. This product definition has been done based on such requirements.
1.3. CONTENTS

This document contains the following sections:

- Section 1. “Introduction”.
- Section 2. “Reference and Applicable Documents” lists of documents that are key to the structure and contents of this document.
- Section 3. "Terms Definitions and Abbreviated Terms" contains the list of terms and definitions that harmonize the nomenclature used providing the clarifications for the correct understanding of the terms.
- Section 4. “Product Overview” describes the main goals for the ESROCOS framework
- Section 5. “Use of ESROCOS in Space Robotics” describes the relation between the different tools used in ESROCOS and the SW development methodology for space, and provides a categorization of the tools according to its associated workflow in the development.
- Section 6. "Environment Preparation Tools describes the characteristics of the tools to be used for ESROCOS installation, the different containers for the framework, and the target development environments.
- Section 7. “Third-party SW Integration Tools” describes the different approaches for the integration of existing robotic developments into ESROCOS
- Section 8. “Robotics Modelling Tools” describes the tool that ESROCOS will include for robotic kinematic modelling.
- Section 9. "TASTE Modelling": describes the capabilities that ESROCOS will have with respect to system’s modelling.
- Section 10. “BIP Modelling Tools”: describes the approach followed in ESROCOS to integrate BIP tools into ESROCOS.
- Section 11. "IMA Modelling": describes the integration of AIR into ESROCOS
- Section 12. “Code Generation Tools”: discusses the target environments for the code generation, and its characteristics.
- Section 13. “Additional SW ASsets”: a specific set of assets are to be developed in the frame of ESROCOS, that are to be used, either during the system architecture and design, or as libraries to be used during runtime. These are commented in this section.
- Section 14. “ESROCOS Overall Architecture” provides the global view of the ESROCOS system.
2. REFERENCE AND APPLICABLE DOCUMENTS

2.1. APPLICABLE DOCUMENTS

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AD.1]</td>
<td>Peraspera: D3.1 Compendium of SRC Activities (for call 1)</td>
<td></td>
</tr>
<tr>
<td>[AD.2]</td>
<td>Guidelines for strategic research cluster on space robotics technologies horizon 2020 space call 2016</td>
<td></td>
</tr>
<tr>
<td>[AD.3]</td>
<td>ESROCOS EUROPEAN SPACE ROBOTICS CONTROL AND OPERATING SYSTEM HORIZON 2020 Proposal: PART B</td>
<td></td>
</tr>
<tr>
<td>[AD.4]</td>
<td>ESROCOS D1.1 Technology Review issue 1.0</td>
<td>19/01/17</td>
</tr>
<tr>
<td>[AD.5]</td>
<td>ESROCOS D1.2 System Requirements issue 1.0</td>
<td>19/01/17</td>
</tr>
<tr>
<td>[AD.6]</td>
<td>ESROCOS D6.2 Dissemination Plan</td>
<td>28/02/17</td>
</tr>
</tbody>
</table>

Table 2-1: Applicable documents

2.2. REFERENCE DOCUMENTS

The following is the set of documents referenced:

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RD. 8]</td>
<td>Behaviour, Interaction, Priority (BIP) <a href="http://www.versimag.imag.fr/Rigorous-Design-of-Component-Based.html">http://www.versimag.imag.fr/Rigorous-Design-of-Component-Based.html</a></td>
<td>-</td>
</tr>
<tr>
<td>[RD.10]</td>
<td>ASN.1 Website: <a href="http://www.itu.int/en/ITU-T/asn1/Pages/asn1_project.aspx">http://www.itu.int/en/ITU-T/asn1/Pages/asn1_project.aspx</a></td>
<td>-</td>
</tr>
<tr>
<td>[RD.11]</td>
<td>AADL Website: <a href="http://www.aadl.info/aadl/currentsite/">http://www.aadl.info/aadl/currentsite/</a></td>
<td>-</td>
</tr>
<tr>
<td>[RD.18]</td>
<td>J. Delange and M. Perrotin “On integration of open-source tools for system validation, example with the TASTE tool-chain” 13th Real-Time Linux Workshop</td>
<td></td>
</tr>
<tr>
<td>[RD.19]</td>
<td><a href="https://www.khronos.org/collada/">https://www.khronos.org/collada/</a></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-2: Reference documents
3. TERMS DEFINITIONS AND ABBREVIATED TERMS

3.1. DEFINITIONS

Concepts and terms used in this document and needing a definition are included in the following table:

Table 3-1: Definitions

<table>
<thead>
<tr>
<th>Concept / Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2. ACRONYMS

Acronyms used in this document and needing a definition are included in the following table:

Table 3-2: Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AADL</td>
<td>Architecture Analysis and Design Language</td>
</tr>
<tr>
<td>AD</td>
<td>Applicable Document</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASN.1</td>
<td>Abstract Syntax Notation One</td>
</tr>
<tr>
<td>BIP</td>
<td>Behaviour, Interaction, Priority</td>
</tr>
<tr>
<td>BIP RTE</td>
<td>BIP Runtime Environment</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>ECSS</td>
<td>European Cooperation for Space Standardization</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>ESROCOS</td>
<td>European Space Robot Control Operating System</td>
</tr>
<tr>
<td>FDIR</td>
<td>Failure Detection, Isolation and Recovery</td>
</tr>
<tr>
<td>GRSL</td>
<td>Geometric Relations Semantics Library</td>
</tr>
<tr>
<td>IMA</td>
<td>Integrated Modular Avionics</td>
</tr>
<tr>
<td>MDA</td>
<td>Model-Driven Architecture</td>
</tr>
<tr>
<td>OROCOS</td>
<td>Open Robot Control Software</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PA</td>
<td>Product Assurance</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PMK</td>
<td>Partition Management Kernel</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>PUS</td>
<td>Packet Utilization Standard</td>
</tr>
<tr>
<td>RAMS</td>
<td>Reliability, Availability, Maintainability and Safety</td>
</tr>
<tr>
<td>RCOS</td>
<td>Robot Control Operating System</td>
</tr>
<tr>
<td>RD</td>
<td>Reference Document</td>
</tr>
<tr>
<td>RDEV</td>
<td>RCOS Development Environment</td>
</tr>
<tr>
<td>RM</td>
<td>Robot Modelling</td>
</tr>
<tr>
<td>ROCK</td>
<td>Robot Construction Kit</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>RTEMS</td>
<td>Real-Time Executive for Multiprocessor Systems</td>
</tr>
<tr>
<td>RTT</td>
<td>Real-Time Toolkit</td>
</tr>
<tr>
<td>SAE</td>
<td>Society of Automotive Engineers</td>
</tr>
<tr>
<td>SARGON</td>
<td>Space Automation and Robotics General Controller</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SPARC</td>
<td>Scalable Processor Architecture</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TASTE</td>
<td>The ASSERT Set of Tools for Engineering</td>
</tr>
<tr>
<td>TBC</td>
<td>To Be Confirmed</td>
</tr>
<tr>
<td>TSP</td>
<td>Time and Space Partitioning</td>
</tr>
<tr>
<td>URDF</td>
<td>Unified Robot Description Format</td>
</tr>
</tbody>
</table>
4. PRODUCT OVERVIEW

The ESROCOS activity is devoted to the design of a Robot Control Operating System (RCOS) based on existing open-source software and on the TASTE toolset [RD. 12], with special care on its RAMS characteristics. In the activity we shall analyse in depth the requirements for an RCOS and what is needed to complement the current TASTE implementation for covering all needed RCOS functionalities. The ESROCOS project has the following objectives:

- **Obj 1. Develop a Space-oriented RCOS:** ESROCOS shall target space development needs by including space-grade RAMS attributes (refer to ESA ECSS standards) and off-line/on-line formal verification, Telemetry and Telecommand (TM/TC) messages and qualification of industrial drivers such as the Controller Area Network (CAN) bus or EtherCAT protocols. Two reference implementations shall be carried out on space representative avionics.

- **Obj 2. Integrate advanced modelling technologies:** ESROCOS shall include complete model-based methodology supporting the design of the individual components as well as the interfaces for their interaction and integration, the verification of the structural and behavioural properties at the system level, and a framework that also provides glue code generation. This approach allows the separation of the model from the target platform, which is a requirement for the reuse of the software in future developments.

- **Obj 3. Focus on the space robotics community:** ESROCOS requirements will be consolidated by actors leading state-of-the-art robotics space missions.

- **Obj 4. Allow integration of complex robotics applications:** ESROCOS shall provide a flexible architecture, following the Time and Space Partitioning [RD.6], which also allows hosting different level of space quality applications over the same on-board computer.

- **Obj 5. Avoid vendor-lock in situations:** The product is to be delivered as open-source code (Mozilla Public License, Apache, MIT, BSD and GPL/LGPL), avoiding proprietary solutions that can have difficulties in being adopted.

- **Obj 6. Leverage on existing assets:** Instead of starting from scratch, ESROCOS shall enhance already existing frameworks (TASTE extended with a robotics components approach inspired by the ROCK middleware [RD. 4]), mature toolsets (source-code versioning, scripting/testing, visualizers/simulators) and libraries (advanced data types, robotics transformations of reference systems, robotic arm kinematics and dynamics, rover locomotion control).

- **Obj 7. Ease the development of robotics systems:** ESROCOS shall be interoperable with other robotics frameworks (e.g. ROCK/ROS 3rd-party libraries and visualizers/simulator) allowing testing their algorithms together with space critical components.

- **Obj 8. Cross-pollinate with non-space solutions and applications:** ESROCOS shall benefit from the experience gathered in developing RCOS for robots in the nuclear environment, with very stringent RAMS requirements.

In the following sections, we will describe how the resulting product will cover these areas.
5. USE OF ESROCOS IN SPACE ROBOTICS

The ESROCOS product will be a framework for the development of robotic applications. This framework will combine together a number of existing and proven assets as well as newly developed software tools and components in order to provide an integrated solution for the development of space robotics systems with demanding RAMS requirements.

This framework will be formed by already existing components (not developed in the frame of ESROCOS) and newly developed components. Both the existing components used in ESROCOS and those components to be developed in the frame of the project will have open-source licenses associated to them to ensure that they can be used by a wide community.

The following figure shows the activities associated to the development of software in the space industry. It has been taken from the European Standards [RD. 17].

![Diagram of software development activities](image)

Figure 1: Activities associated to SW development, described in ECSS [RD. 17]

The ESROCOS product design has been done having in mind these activities. In the following sections we will describe the use of the different tools provided by the ESROCOS framework in different workflows, as well as the impact in the space development activities.

The ESROCOS framework will be formed by a set of tools to support these activities. Each of these tools will be oriented to cover a specific need in a different areas, and therefore we can categorize the different tools based on the workflow for which they will be applied, namely:

- **Robotic Environment preparation tools:** any RCOS requires developers to setup the corresponding framework. These tools are discussed under section 6.
- **Third-party SW integration tools:** Most of the existing SW robotic developments rely on existing RCOS. Among them, some of the most important ones are ROS and ROCK. ESROCOS will provide a set of tools to port existing developments done in these frameworks within ESROCOS. Also tools to interconnect to existing robotic development (ROS, ROCK) will be provided. This is described in section 7.
- **Architecture and Design tools:** ESROCOS will provide a new paradigm based on the Model-Driven Architecture (MDA) evolutionary, architectural design. ESROCOS will
provide a tight cohesion between Architecture, Design and implementation activities by using the MDA paradigm, in which part of the code is automatically generated from abstract, human-elaborated modelling diagrams.

- **Advanced Modelling tools:** The ESROCOS solution is therefore based on the ESA TASTE toolset [RD. 16][RD. 14]. But the existing capabilities provided by TASTE will be enhanced in ESROCOS in the following way:
  - ESROCOS will complement TASTE with a Robotic Kinematic modelling tool. Section 8. of this document describes the foreseen use of the ESROCOS modelling tools in the frame of ESROCOS
  - ESROCOS will benefit from existing TASTE capabilities. This is explained in Section 9.
  - In addition, TASTE will be complemented by BIP [RD. 8][RD. 12]. BIP is a well known tool for the development of correct by construction methods, developed by VERIMAG, BIP will be used for 1) Schedulability analysis and 2) guarantee, by design and during run-time, that some predefined constraints are being preserved. This is explained in Section 10.
  - ESROCOS will also extend the capabilities of TASTE with respect to modelling and deploying IMA partitions [RD.6]. This is discussed in Section 11.

- **Code Generation tools:** In ESROCOS, there are three different target environments for code generation: space application (SPARC/RTEMS), high fidelity and laboratory environment (these last using Intel/Linux). From the different models created with TASTE, code for each platform can be generated. This is a current capability for TASTE, that will be enhanced with the addition of new drivers. Code generation capabilities are described under Section 12.

- **SW Assets:** Together with the development tools, ESROCOS will provide a set of additional assets that can be used in a robotic application for the space development, such as coverage analysis tools, schedulability analysis tools, robotic simulators, etc. These are detailed in Section 13.

In the following sections, we provide an introduction to the tools as well as its intended usage.
6. ENVIRONMENT PREPARATION TOOLS

The ESROCOS software relies on the TASTE toolset, which must be installed together with the other components. The installation of ESROCOS's TASTE will be covered by the toolset’s documentation, which will be downloaded from the Internet. The development environment for ESROCOS requires an x86 system running either Linux 32-bit or Linux 64-bit. Linux-32 bit will be the recommended environment. The product will be provided in two different formats:

- Either in the form of a virtual Linux machine distributed with the toolset as host system for ESROCOS.
- Or as a set of scripts that will allow the installation of ESROCOS from the ground up in a regular Linux system (32-bit or 64-bit).

The ESROCOS setup scripts will manage dependencies and the installation of ESROCOS itself. The scripts will make use of additional tools, such as Ruby and Git, that will be required for downloading and installing ESROCOS.

As indicated in the "Communications and Outreach Manual" [AD.6] the primary source of information for public will be the project’s website.

Currently it is reachable under the URL http://www.h2020-esrocos.eu/. Figure 4-1 shows a screenshot of the front page of the website. The public website contains updated information about the project and its progress. This includes:

- General project objectives
- Information about the consortium
- Project schedule include milestones
- Information about deliverables
- Information about publications
- A news section, where for example important events are mentioned
- A private area

Once the ESROCOS product will be available, a link will be provided from this web in order to download the framework from this website.

The installation of ESROCOS and the installation or upgrade of TASTE will require a connection to different Internet resources. In case that an organization uses a network proxy, the host Linux system must be configured to use it. For instance, using the apt-get or wget commands used by the different scripts for the installation may require specific proxy settings to be defined in the user’s account and the root account of the system. Details will be provided in a future ESROCOS user's manual document.
Project Objectives

The ESA/MARINA project is focused on the development of an advanced control and operating system (ESROCOS) that aims to provide a robust, adaptable, and scalable solution for space missions. The main objectives of the ESROCOS project are to:

1. Develop an open-source framework that can be easily integrated into various mission control systems.
2. Provide a modular design that allows for easy reconfiguration and scalability.
3. Ensure high reliability, availability, maintainability, and safety (RAMS) properties.
4. Support real-time operation and control in the space environment.
5. Facilitate efficient integration with existing ground systems.

The ESROCOS system will be designed to support a wide range of space missions, from small satellite operations to large space telescopes. By providing a flexible and adaptable platform, ESROCOS will enable mission operators to quickly adapt to changing mission requirements and reduce development costs.

Figure 2: Public website of ESROCOS
7. THIRD-PARTY SW INTEGRATION TOOLS

As mentioned before, ROCK and ROS are robotic operating systems that have a complete set of features and are being used worldwide. In particular, ROCK has been used by ESA in some of its developments, and therefore it is of particular importance for the space environment. The integration of ESROCOS with these frameworks will cover the following aspects:

- Generating Rock or ROS SW components from TASTE model
- Porting existing developments in ROCK or ROS into ESROCOS
- Interoperability: allowing ESROCOS-generated images to operate in combination with other ROS and ROCK binaries

TASTE is a MDA tool aimed to provide a Platform Independent Model (PIM), from which code for different “platform specific models” (PSM) can be generated. From that point of view, different transformations are possible. The following figure shows possible transformations from the ESROCOS Platform independent model into different platform specific models. So for instance in Figure 3, the transformation 1 could provide glue code for a future OROCOS RTT (ROCK) transformation, while the transformation 2 will provide gluecode generation for C or Ada into SPARC/RTEMS (space environment), and transformation 3 will provide code for a Linux (Laboratory environment) machine.

![Figure 3: Example of TASTE Transformations and Code Generation](image-url)
With respect to the porting of existing elements, it would be sufficient to provide the opposite transformation, that is, the generation of parts of a PIM model from a PSM. This "reverse" transformation cannot be complete in most cases, since the high level model usually requires detailed information not present in the platform-specific models.

Nevertheless, some already existing tools, like the rock2taste tool, a tool developed in the frame of the SARGON project, that is able to generate parts of the TASTE models from existing sources and components of a existing development in ROCK, will be part of ESROCOS as well, providing this capability. A similar approach will be followed for the ROS integration.

The third possibility for ROS/ROCK integration tools is the generation of a "bridge" among architectures. This "bridge" will make possible the interaction of ESROCOS and ROCK or ROS executables during runtime; so for instance a ROS node in Linux could communicate with a TASTE component in a space qualified processor (SPARC/RTEMs) by using this bridge.

Details on the exact number and scope of the toolset are under discussion. However, ESROCOS will provide a solution for these three different approaches.
8. ROBOTICS MODELLING TOOLS

ESROCOS aims to extend the modelling capabilities of TASTE for robotics components adapting the use of the AADL to the particular robotics scenario. Existing tools like Geometric Relations Semantics Library (GRSL) and Kinematics and Dynamics Library (KDL) allow to model kinematic chains, such as robots, biomechanical human models, computer-animated figures, machine tools, etc (see Figure 4). They provide class libraries for geometrical objects and their motion specification and interpolation. The GRSL supports checking KDL transformations for semantic correctness. The TASTE meta-model could therefore be extended to include the capabilities needed for this robot modelling and checking the semantics of geometric frames and transformations between components. AADL allows for such extensions using the Domain Data Modelling Annex SAE AS5506/2.

A Robotic Kinematic Modeller tool will come as part of the ESROCOS framework. The Robot kinematic modeller will generate the kinematic model of the robot using the robotics modelling tool. This model will include the joints, links and transmission of the robot, the geometry of the elements, the axis of rotation, etc. For this purpose, it will use a set of models, namely:

- The constraints model provides a consistent set of restrictions to be preserved
- The Formal kinematics model contains all the models defined using the RM tool
- The geometric model contains links and joints of the robot, as well as connection types
- The mechanical model contains attachment frames, transmission and actuator types

**Figure 4: Geometric primitives useful to define the position, orientation, pose, linear velocity, angular velocity, and twist of body C with respect to body D (left) and KDL robotic arm modelling (right)**
And generates the following outputs:

- The Dynamic model contains actual force and/or acceleration inputs in Cartesian and/or joint space; relative weighing or prioritization of the inputs.
- The formal kinematics model contains all the models defined using the RM tool.

The robotic kinematic modeler will use the Robotic modelling tools of ESROCOS to generate the formal model and the instantaneous dynamic data.

The Robotic modelling toolset will be composed of several applications (TBC):

- A first application, a semantic modeler, will be used by the Robotic Modeller to build the Semantic models of kinematic chain. It will accept as inputs a list of robot components (links and joints), with connection types, attachment frames, transmission and actuation types and produce as outputs a formal model (structural and behavioural) of the composite chain, on which geometric, dynamic and logical queries can be answered. The model produced is semantically complete, in that it contains information about physical units, mathematical representations, and numerical representations. The level of refinement of such tool has not been defined yet, it can be basic, or more advanced, e.g., having the capability to work in in interactive, iterative mode which provides a list of not-yet-complete semantic constructions, and with suggestions about how to complete the model. The envisaged scope of the modelling contains all lumped-parameter robot "arms", "mobile platforms" and "grippers".

- A second application will be the Hybrid dynamics instantaneous motion solver. This is the core algorithm to compute the motion behaviour of all of the above-mentioned robot systems. It will receive as inputs a formal model of Item 1.; actual force and/or acceleration inputs in Cartesian and/or joint space; relative weighing or prioritization of the inputs, and produce as outputs joint torques required to realise, instantaneously, the input specification.

- A third application, a "motion specification" with symbolically verifiable constraints. While Item 2. provides a purely numerical solver, this activity adds a (selection of) symbolically defined primitives, which can (i) be put into the symbolic representations of BIP models, and (ii) be translated automatically into concrete numerical representations for the hybrid solver. Possible examples are:
  - "parallel" and "orthogonal" constraints on symbolically identified parts of the robots or their environment;
  - "mereo-topological" constraints such as "in front of", "above", "near to";
  - "spatio-temporal" dependencies such as "left arm has motion priority over right arm".

- This item requires a close cooperation with the BIP activities, and the exact contents of this activity is driven by a project-wide focus.

- A fourth application will perform C-code generation of kinematics and dynamics solver, with guaranteed constraints on (statically allocated) memory and (discretely checked, runtime "anytime solver") worst case execution time.

It is expected that these tools will be able to produce files in the format required for interactive 3D applications, using standards such as Collada [RD. 19].
9. TASTE MODELLING

The TASTE Model Generation is the process by which a user produces a TASTE model based on the requirements and the architecture.

As indicated in [RD. 7] A typical TASTE Application is composed of the following layers:

1. The Application layer
2. The Glue Layer
3. The Middleware
4. The OS with BSP support

The application layer is the application code for the application functions to be performed by the system. This is the code of the user, written either via application models (SDL, Matlab/Simulink) or via a regular programming language (Ada, C...). The application layer is basically a set of functions - written by the user - that interact with the automatically generated functions in the application skeletons that are generated from a set of views.

TASTE is an open source framework that allows the development of embedded, real-time systems. It relies on key technologies such as standardized modelling languages (e.g., ASN.1 and AADL), code generators and real-time systems that allow generating the suitable skeletons and the production of the system executable. The designers implement their embedded systems using a set of views, abstracting the user from the implementation details of the underlying platform (e.g., operating system, drivers) and guaranteeing the fulfilment of real-time properties.

![Diagram](image)

Figure 5: TASTE interface view (top) and deployment view (bottom)
10. BIP MODELLING TOOLS

Within the space context all missions “must” be successful due to its one-single-shot nature. Moreover it is very difficult to test the software in its real operational environment (rockets, spacecraft) due to reasons such as low risk profile of engineering, the need of full controllability or strict software engineering and product assurance (PA) standards. Thus, there is a real need of securing all possible hazards by means of redundancies and robust safe modes. The software development process must minimize the risk to activate a software design error and prevent any software failure by applying different measurements:

- All behaviour is tested with full coverage of source code and traceability against requirements.
- Inexistence of dead code and dynamic memory allocation.
- Deterministic and predictable real-time behaviour.

Therefore there is a real need of applying exigent RAMS attributes to the SW development process:

- Reliability: when the software works, it works well (ECSS-Q-ST-30)
- Availability: the software always replies to request, even if it is failing
- Maintainability: the software can be modified at reasonable effort
- Safety: a software error is not going to damage any asset or person (ECSS-Q-ST-40)

In order to satisfy such RAMS requirements typically the space software validation process include expensive test campaigns to cover all system requirements (functional coverage), the source code (structural coverage) and real-time behaviour. ESROCOS will allow to reduce such verification and validation process by:

- Replacing the huge amount of testing needed by verification at design level (schedulability analysis and correct-by-construction techniques)
- Separating the concerns in the architecture (real-time, communication, hardware access, etc.) such as the testing of each concern is easier.
- Providing full visibility and understanding of the code (no dead code, no malloc)

Architectures are essential for mastering the complexity of systems and to facilitate their analysis and evolution. They allow a separation between the detailed behaviour of the components and their overall coordination. Coordination is usually expressed by using constraints that define possible interactions between components.

BIP supports a component construction methodology based on the thesis that components are obtained as the superposition of three layers. The lower layer describes behavior. The intermediate layer includes a set of connectors describing the interactions between transitions of the behavior. The upper layer is a set of priority rules describing scheduling policies for interactions.

The RAMS properties of a ESROCOS design will be verified during the validation phase using the BIP framework. Our approach is based on the previous experience with the TASTE2BIP tool developed under the ESA-funded project MoSaTT-CMP and SARGON projects to verify space-quality applications.

The main objective of the TASTE2BIP tool is to translate both the application and scheduling policy into a VERIMAG’s formal mixed automata-based/procedural language called BIP (which stands for Behaviour Interactions Priorities). A special feature of the BIP framework is the co-programming of a scheduling policy together with application. They are expressed as a network of interacting timed automata. This approach gives a two-sided benefit. On the one hand, it provides rigorous formal semantics of system evolution in physical time, which facilitates formal analysis and proofs of system schedulability. On the other hand, BIP serves as a programming language with code generation and runtime environment (BIP RTE), which allows executing the BIP directly on the embedded platform. This realizes a principle “what you verify is what you execute”, which facilitates the correct-by-construction argument of system design.
11. IMA MODELLING

Resolving the challenges associated with the adoption of multi-core CPUs in safety critical systems can be a daunting task. A solution is Time and Space Partitioning (TSP), also known as Integrated Modular Avionics (IMA) [RD.6], where real-time applications are isolated from each other in terms of time and space.

IMA enables multiple unrelated applications, with different criticalities, to share the same computing platform without interference by applying robust partitioning. Modular architectures provide a shared resource pool, composed of processing, power, communications, and I/O modules, that is partitioned among multiple functions. The partitioning keeps applications from inadvertently influencing each other by enforcing strict separation segregating computing resources in space and time. TSP is accomplished by defining application memory areas with strict access rights and running those applications according to a static cyclic schedule, ensuring that only one application/functionality can execute at a time and for a predefined amount of time. The IMA concept allows applications to have their software life cycle developed independently from the remaining system. Validation and certification processes can be applied to partitions according to their criticality level.

Figure 6 illustrates the two level scheduling policy implemented by modular architectures. The processor time is distributed among partitions (Partition Schedule) in a static and fixed schedule while processes are allowed a less strict scheduling (Processes Schedule). The concept is successfully applied in the aeronautical industry (e.g. Boeing 787, Airbus 380).

More recently, in the aerospace domain, IMA is being studied by the European Space industry in the scope of several activities. The groundwork has been done in the AIR and AIR II studies and the AMOBA project, promoted by GMV. These projects focused on the technical feasibility of partitioning on space-typical hardware, namely the LEON2 and LEON3 processor. With IMA for Space, the state-of-the-art of IMA in aviation and other industries was successfully transferred to the space domain.

AIR follows a two level architecture; the lower level is composed by a software hypervisor that segregates computing resources between partitions, while a second level, the application level, is composed by system’s or user’s applications running in an isolated environment (a partition). A partition is the basic unit to which the segregated resources are allocated. The hypervisor, named partition management kernel (PMK), is responsible for the robust isolation of the applications and for implementing the static CPU allocation schedule.

At the application level, each partition can run a different para-virtualized partition operating system or no operating system at all, acting, therefore, just as a bare executive. Currently several real-time operating systems are supported (RTEMS, ARINC 653, Bare C-executive). To deal with Input/Output, AIR provides a configurable out-of-the-box I/O server, an I/O partition. This dedicated partition maps ports to addresses in the data bus or network, making I/O transparent for the applications. The I/O partition already provides drivers for: Ethernet with Deterministic IP-Stack (GRETH), MIL-STD-1553, Space-Wire (GRSPW, GRSPW2) and RS-232 (APBUART).

In terms of multi-core, AIR is currently multi-core aware, supporting the NGMP processor [IMA.5]. AIR provides a supervised AMP mapping, extending the IMA partition schedule to support more than one core simultaneously. This solution encompasses a cyclic schedule per core.
with the possibility of defining multiple module schedules. Schedules support idle execution windows. As a result, bus contention can be removed for critical applications with small impact on overall performance. In this allocation schema, the partitions do not need to be multi-core aware, as they execute only in a single core at a given moment. However, more than one partition is executed simultaneously according to the developer-defined schedule. In addition, it is possible to allocate more than one processor core to a given partition.

Figure 7 depicts an example of scheduling multi-core capabilities. In it, we can see a partition schedule per core with single (P4) and multicore partitions (P1, P2 and P5) and idle execution slots to minimize the interference to Partitions 3 and 5.

The figure can also be used to illustrate the mixed criticality capabilities of AIR, assuming P5 and P3 partitions as critical applications, where through the presented configured schedule it is ensured their execution has no interference in time or memory.

Under the ESROCOS context, the AIR hypervisor will become an enabler for these capabilities:

- **Transfer of the component/model base realization to target implementation** through the association of components to different partitions. Therefore critical components with hard-real time requirements would be in specific partitions running on a space-qualified RTOS, where they run together with other less critical functionality on a second partition (allowing to offer the sand-box approach where the hypervisor can reboot the second partition in case of failure). The respective partition schedule would then allocate the appropriate resources in terms of time, exclusivity and processor cores to facilitate the real time tasks targets.

- **Support of the robotics component life-cycle.** IMA further enables this life-cycle with the definition of multiple module schedules (MMS), where it activates the partitions (components) relevant for a given life-cycle state and optimize the time/core/memory resources for the active partitions.

- **FDIR functionality,** as AIR also includes a configurable health monitor.

The TASTE toolset already includes some level of integration to the IMA approach, allowing to model the deployment of designed software. We foresee to upgrade the support of multi-core configurations by specifying the association of processor core to partitions. The remainder of mixed-criticality related activities will then incise on the improvement of AIR to support the best hardware applicable to ESROCOS; namely, the support to the selected hardware architectures (e.g.: Linux and RTEMS) and the avionics bus drivers (e.g. CAN and Ethernet buses).
12. CODE GENERATION TOOLS

ESROCOS is a multi-platform RCOS: any development done using the ESROCOS RCOS can be tailored (by using the TASTE Deployment View) to generate code for three different environments:

- The **laboratory environment**: is a typical environment that corresponds to a SW criticality E; the target operating system is Linux 32-bits, and there are no restrictions with respect to the use of C++ dynamic memory allocation, or the use of external libraries.

- The **high-quality environment** is targeting a LEON platform using the RTEMS operating system. Applications developed in this environment correspond to SW criticality C or D; they shall be implemented using either Ada or C, or a subset of C++ suitable for real-time applications.

- Finally, the **space qualified environment** is targeting space developments in which a higher SW criticality is required (either A or B): In this case there is a strict control of the SW to be used, dynamic memory allocation is not allowed, and the system must comply to the rules for the development of the highly critical SW.

For the high quality and space qualified runtime environment, a specific runtime environment for a space-qualified SPARC board will be taken as reference. Nevertheless, the system will be able to generate code for a large bunch of different LEON/RTEMS platforms (being the compatibility of the code the same as the one provided for any TASTE development).
13. ADDITIONAL SW ASSETS

In addition to the previously commented assets, ESROCOS will include a set of SW assets, that could be used during the modelling of the system and during runtime. The following is an initial list, to be completed in further versions of the project:

- TASTE already provides a set of bus drivers (e.g., Spacewire, serial, 1553) but it is missing support for the CAN bus or EtherCAT. In order to be able to develop any robotic application, it is necessary to develop such assets. CAN and EtherCAT drivers for TASTE will come as part of the ESROCOS.

- Space applications use the PUS standard [RD.26] [RD.27] to capture and command the status and health of the on-board units. The ESROCOS Consortium will generate TASTE components and libraries to support the most suitable PUS services for the RCOS needs, like:
  - PUS Service 1: Telecommand Verification Service
  - PUS Service 3: HouseKeeping
  - PUS Service 8: Function management service (for mission specific functions)
  - Monitoring and reporting: Event reporting (PUS Service 5), on-board monitoring (PUS Service 12) and event-action (PUS Service 19).
  - PUS Service 23: File Management for the new ECSS standard (ECSS-E-ST-70-41C draft1); although the final version of the standard is not published, a specific services has been introduced for file management, it is proposed to use this service for the activity, and additional operations could be included if needed.
  - Any Additional PUS Services that might be found of interest.

- Common data types are basic types in order to exchange relevant information in RCOS. Those types have a dual purpose. Firstly they provide the API interface to exchange data between the framework (e.g. RCOS) and the library (e.g. the algorithms). Secondly, they are data structures to serialize and send through the communication layer provided by RCOS. A set of robotics data types related to geometry, sensor data, actuator control, navigation will be provided in ASN.1 format in ESROCOS, together with a set of conversion and utility functions. TASTE will generate the necessary serialization functions for data transport.

- Logging functions: A centralized logging function is very useful during the prototyping and development of a robotic system, and therefore it has been identified as a key component of the ESROCOS infrastructure for the laboratory and high-reliability environments. For the space environment, the PUS TM functions should be used instead. The logging function allows the programmer to issue messages that help analysing the internal behaviour of the software of the different functions of the robotic system. Messages are identified, timestamped and stored in a centralized file for off-line analysis. ESROCOS will include a Centralized logging component.

- Component Lifecycle: the objective of a component life-cycle is to allow control of a given component from an external source. The software in charge of controlling the lifecycle of the component allows to either locally or remotely control the start-up, configuration and deactivation of the component. In addition, the software in charge of the lifecycle allows at the same time the monitorization of the component from external SW. ESROCOS will include a component lifecycle model, that can be tailored to fulfil specific needs, or be used in a generic way.
14. ESROCOS OVERALL ARCHITECTURE

The overall architecture of the ESROCOS system, outlined in the ESROCOS proposal [AD.3], is outlined in Figure 8, and emerges as the solution from the ESROCOS software technical requirements. The ESROCOS framework will be composed of existing SW as well as newly developed SW that will fulfill the goal of the functional blocks of the figure.

As we can see in the figure, the innovation behind ESROCOS lies in the following main elements:

- ESROCOS provides a robotic modelling tool (1), able to capture semantic information related to the kinematics, that can be used in models.
- Integrated with existing tools for visualization, simulation and scripting, to port existing robotic applications to ESROCOS, or to interconnect ESROCOS developments with other robotic systems (11).
- Providing a system for deployment, configuration and continuous system integration of the ESROCOS toolset (12).
- Ease the use of robotics developers by including basic robotics libraries (4), ensuring interoperability with 3D robotics viewers, simulators and third-party libraries, guarantee advanced logging capabilities (5) as well as identifying advanced data types and a set of robotic libraries (2,4) (closing the gap between ASN.1 and robotics).
- ESROCOS will include a Robotic State Machine for FDIR, that can be used by TASTE components (3).
- Providing a library for PUS Services as a TASTE component, to be used both for Space and laboratory applications (6).
- ESROCOS uses a Model-driven approach for the Middleware, based on TASTE (7) with extended, robotic-specific modelling semantics, allowing the design of a Platform Independent Model, thereby respecting the fact that space hardware varies depending on the chosen vendor.
- Answering the needs of future robotic needs, by being able to interact across multiple components with a mixed-criticality approach, following correct-by-construction methods, by integrating TASTE with the BIP toolset (8).
- Allowing the use of IMA in TASTE designs, and having an IMA implementation (9).
- Developing SW assets specifically targeting the space application needs, such as TASTE drivers for Ethercat and CAN bus, filesystem services (10).
- Defining different target environments: a laboratory environment, in which we have no criticality, a high quality environment, targeting environments of higher SW criticality, and finally a space-grade system, that complies with space processors and avionics drivers, command and telemetry access services as well as following SW development space-standards (13).