SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) is an ESA project (Contract ESTEC 14898/01/NL/JA) conducted by a consortium led by EADS Astrium SAS with AXLOG Ingénierie and LAAS-CNRS. For more information please contact:

Maria Hernek
ESTEC. Keplerlaan 1. PO Box 299
2200 AG Noordwijk ZH – The Netherlands
Tel: +31 (0) 71 565 5821. Fax: +31 (0) 71 565 4798
e-mail: maria.hernek@esa.int

Jean-Paul Blanquart
EADS Astrium
31, avenue des cosmonautes
F-31 402 Toulouse Cedex 4, France
Tel: +33 5 62 19 69 56. Fax: +33 5 62 19 71 58
e-mail: jean-paul.blanquart@astrium.eads.net

Jean-Clair Poncet
AXLOG Ingénierie
19-21, rue du 8 mai 1945
F-94 110 Arcueil, France
Tel: +33 1 41 24 31 00. Fax: +33 1 41 24 07 36
e-mail: jean-clair.poncet@axlog.fr

Felix Ingrand
LAAS-CNRS
7, avenue du colonel Roche
F-31 077 Toulouse Cedex 4, France
Tel: +33 5 61 33 78 04. Fax: +33 5 61 33 64 55
e-mail: felix@laas.fr
The project objectives were to investigate dedicated software product assurance measures to support autonomous functions both for nominal spacecraft operations and for fault detection, identification and recovery management, i.e., how to ensure safety and dependability of autonomous space software and especially of software in charge of autonomous functions dedicated to the spacecraft safety and dependability management. Special attention was put on software product assurance for advanced autonomy techniques (artificial intelligence, self learning techniques, etc.).

The project was split in two phases. The first phase investigated the lessons learnt from autonomous non-space applications, the software product assurance requirements and then methods, tools and procedures, for autonomous space systems. Special autonomy software safety aspects were then investigated and an implementation plan was elaborated for the second phase. The main results highlighted on the one hand the necessity to cope with residual faults during execution (fault tolerance), and on the other hand the interest of intensive simulation-based testing before actual operations. The second phase was dedicated to the definition of software functions (on-board and in the ground system) for the safety of spacecraft with autonomy, and to their implementation and assessment through a pilot application. Two components were developed and experimented, in line with the results of phase 1:

- An on-board component (the “safety bag”) for monitoring on-line of safety properties;
- A ground based component (the “plausibility checker”) for complementary extensive validation of interpreted procedures.

**Deliverables:**

**Phase 1:**
- Technical Note 1 “Lessons learned from autonomous non-space applications”
- Technical Note 2 “Software product assurance requirements for autonomous space applications”
- Technical Note 3 “Software product assurance methodology for the development of autonomous space systems”
- Technical Note 4 “How to ensure safe decisions in autonomous space systems: candidate techniques for autonomy on-board spacecraft”
- Technical Note 5 “Plan for implementing assurance software for autonomy functions”
- Phase 1 final presentation slides

**Phase 2:**
- Technical Note 6: “Experimentation of SPAAS reusable components”
- Technical Note 7: “Identification, collection and assessment of certification requirements”
- Software data packages (on-board: “safety bag” and ground: “plausibility checker”): code, associated files and documentation
- Final presentation slides
- Final Report (this document), abstract

The work described in this report was done under ESA contract. Responsibility for the contents resides in the author or organisation that prepared it.

Name of author: Jean-Paul Blanquart (EADS Astrium SAS)
# Table of Contents

1. Introduction .......................................................................................................................... 5
   1.1 Scope .................................................................................................................................. 5
      1.1.1 Scope of the Project .................................................................................................... 5
      1.1.2 Scope of the Document .............................................................................................. 5
   1.2 Related Documentation and Products .............................................................................. 6
      1.2.1 Reference and Applicable Documents ......................................................................... 6
      1.2.2 SPAAS delivered management items ........................................................................... 7
      1.2.3 SPAAS delivered technical items ................................................................................ 7
      1.2.4 Published papers ......................................................................................................... 8
   1.3 Definition of Terms and Acronyms .................................................................................... 9
      1.3.1 Definition of Terms .................................................................................................... 9
      1.3.2 Acronyms and Abbreviations ..................................................................................... 9

2. Project Organisation ............................................................................................................. 11
   2.1 Objectives ......................................................................................................................... 11
   2.2 Consortium ....................................................................................................................... 11
   2.3 Study Logic ....................................................................................................................... 12
   2.4 Project Outputs ................................................................................................................ 14
      2.4.1 Technical Notes ......................................................................................................... 14
      2.4.2 Software .................................................................................................................... 14
      2.4.3 Dissemination, presentations, published papers ....................................................... 14

3. Synthesis of results ............................................................................................................ 15
   3.1 Investigations on software autonomy and dependability .................................................. 15
      3.1.1 Autonomous non-space systems ............................................................................... 15
      3.1.1.1 Survey of standards and studies ............................................................................ 15
      3.1.1.2 Airborne autonomous systems .............................................................................. 16
      3.1.1.3 Waterborne autonomous systems ......................................................................... 16
      3.1.1.4 Terrestrial autonomous systems .......................................................................... 17
      3.1.1.5 Product assurance for AI-based autonomous systems ........................................ 17
      3.1.2 Requirements for on-board autonomy software ......................................................... 18
      3.1.3 Methods and techniques for on-board autonomy software ........................................ 19
         3.1.3.1 Autonomous space systems ............................................................................... 19
         3.1.3.2 Software product assurance and methods .............................................................. 20
         3.1.3.3 Product assurance software for autonomous space systems ................................ 21
      3.1.4 Advanced software technologies for autonomy and safety ......................................... 22
         3.1.4.1 Space autonomy functions ............................................................................... 22
         3.1.4.2 Knowledge representation ..................................................................................... 22
         3.1.4.3 Techniques for autonomous component implementation .................................. 23
         3.1.4.4 Architecture for autonomous systems .................................................................. 24
         3.1.4.5 Synthesis ............................................................................................................... 24
      3.1.5 Plans for autonomy and safety software components .................................................. 25

3.2 Software components for safe autonomous on-board systems .......................................... 26
   3.2.1 Safety monitoring: the “safety bag” (on-board) ............................................................. 26
   3.2.2 Interpreted procedures validation: the “plausibility checker” (ground) ......................... 29

3.3 Synthesis and recommendations ....................................................................................... 32
List of Figures

Figure 1 — SPAAS Consortium .................................................................................................... 11
Figure 2 — SPAAS Phase 1 Study Logic .................................................................................... 12
Figure 3 — SPAAS Phase 2 Study Logic .................................................................................... 13
Figure 4 — Safety software components in autonomous spacecraft system ............................... 25
Figure 5 — Safety bag architecture and situation ......................................................................... 26
Figure 6 — Safety bag principle (1) ............................................................................................. 27
Figure 7 — Safety bag principle (2) ............................................................................................. 27
Figure 8 — Plausibility checker architecture and situation ............................................................. 29
Figure 9 — Integrated architecture for plausibility checker experimentation .............................. 30
Figure 10 — Plausibility Checker environment based on JSwat (1/2) ............................................. 31
Figure 11 — Plausibility Checker environment based on JSwat (2/2) .......................................... 31

List of Tables

Table 1 — SPAAS Technical Notes .......................................................................................... 14
Table 2 — SPAAS Software .................................................................................................... 14
Table 3 — SPAAS Presentations .............................................................................................. 14
Table 4 — Published Papers .................................................................................................... 14
1 Introduction

1.1 Scope

1.1.1 Scope of the Project

SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) is an ESA project (contract ESTEC 14898/01/NL/JA). The project objectives were to investigate dedicated software product assurance measures to support autonomous functions both for nominal spacecraft operations and for fault detection, identification and recovery management, i.e. how to ensure safety and dependability of autonomous space software and especially of software in charge of autonomous functions dedicated to the spacecraft safety and dependability management. Special attention was put on software product assurance for advanced autonomy techniques (artificial intelligence, self learning techniques, etc.).

The project was split in two phases.

The first phase investigated the lessons learnt from autonomous non-space applications, the software product assurance requirements and then methods, tools and procedures, for autonomous space systems. Special autonomy software safety aspects were then investigated and an implementation plan was elaborated for the second phase.

The second phase was dedicated to the definition of software functions (on-board and in the ground system) for the safety of spacecraft with autonomy, and to their implementation and assessment through a pilot application.

1.1.2 Scope of the Document

This document is the Final Report of the SPAAS project. It covers the whole project (phase 1 and phase 2) and gives an overview of the project organisation, structure and main findings, outcomes and recommendations.

This introduction provides:

- The scope of the project and of this document (this section),
- The list of related documentation (SPAAS reference and applicable documents, list of SPAAS management documents, SPAAS technical documents and delivered products, and list of published SPAAS related papers),
- The terminology and acronyms used in this document.

After this introduction, this Final Report is split in 2 chapters.

Chapter 2 describes the SPAAS project from an organisational viewpoint:

- Objectives of the project,
- Consortium information,
- Study Logic,
- Project Outputs: technical notes, software products, dissemination, presentations and published papers.

Chapter 3 describes the project from a technical viewpoint, providing a synthesis of the activities, rationale and main findings and results of each one of the project tasks. The chapter 3 concludes with a section providing a synthetic summary of the main project results and recommendations.
1.2 Related Documentation and Products

### 1.2.1 Reference and Applicable Documents


**[ST 08]** "Software Development and Documentation", MIL-STD-498, Department of Defense, USA.


**[RD 03]** "Fault Tolerant On-Board Software (FTOBS)", ESTEC TRP Study Contract 91.1WD.10 ESTEC.


**[RD 05]** "Definition of Space Software Classes", PASCON Work Order 6 CCN1, ESA/ESTEC Contract n°10662/93/NL/NB.
1.2.2 SPAAS delivered management items


[FRP] SPAAS Final Report, April 21, 2004 (this document).


[MPR] SPAAS Monthly Progress Reports.

1.2.3 SPAAS delivered technical items

[SPAAS PS1] Presentation Slides of the SPAAS Final Phase 1 Presentation, ESTEC, Noordwijk, March 25, 2002.


[SPAAS SWB] “Software data package of the SPAAS on-board component”, EADS Astrium and Axlog, SPAAS/SWB issue 1.0, February 13, 2004; composed of:


1.2.4 Published papers


1.3 Definition of Terms and Acronyms

1.3.1 Definition of Terms

A consistent terminology has been used in the project documents, introduced and discussed in the first Technical Note [SPAAS TN1], from which is especially recalled the adopted definitions for autonomous systems. A dictionary definition of “autonomous” is “acting independently or having the freedom to do so”. This broad definition can cover a wide variety of technologies, going from any programmed open- or closed-loop controller to systems using artificial intelligence (AI) techniques for autonomous decision-making. Here we choose to distinguish autonomous systems using knowledge-based approaches from those that do not (we refer to the latter as “automated systems”). We therefore adopt the following definitions:

**Autonomous system:** An autonomous system is defined here to be technical system that can deliver its service without human intervention. This covers a spectrum of autonomous system types, from systems automatically executing scripts to fully automatic systems executing procedures that are planned and selected according to the current system context and knowledge.

**AI-based system:** A system based on technologies that emulate human cognitive processes. These technologies include, for instance, model-based reasoning, case-based reasoning, neural networks, probabilistic networks, planning, etc.

**Automated system:** A classic autonomous system, not using an AI-based approach.

In addition a set of tasks dedicated to the assessment of the developed SPAAS software components was based on material provided in the former ESTEC project SPEC and especially the Technical Note TN3 on quality model requirements [RD04]. For convenience the terminology adopted here is directly related to the SPEC project, especially the words “certification” and “metrics” which may differ from definitions in other contexts.

**Certification:** product assessment approach based on a tailoring, for a given product, of the quality model proposed in [RD04], with an agreement between the customer and the supplier on the tailoring as well as on the allocation to the supplier, the customer or a third party of the evaluation and assessment of the items of the quality model (metrics, see below).

**Metric:** any one of the lowest level indicators defined in the quality model of [RD04] associated to its evaluation method.

1.3.2 Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3T</td>
<td>Three Tier architecture</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>C</td>
<td>Programming language</td>
</tr>
<tr>
<td>CENELEC</td>
<td>Comité Européen de Normalisation Électrotechnique</td>
</tr>
<tr>
<td>CLISP</td>
<td>Common LISP Implementation (open source)</td>
</tr>
<tr>
<td>CNRS</td>
<td>French National Center for Scientific Research</td>
</tr>
<tr>
<td>DHS</td>
<td>Data Handling System</td>
</tr>
<tr>
<td>DHS32</td>
<td>DHS for 32 bit processor (developed by EADS Astrium)</td>
</tr>
<tr>
<td>ECSS</td>
<td>European Cooperation for Space Standardisation</td>
</tr>
<tr>
<td>ERC32</td>
<td>Sparc V7 32 bit processor hardened for Space on-board applications</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>ESTEC</td>
<td>European Space Research and Technology Centre</td>
</tr>
</tbody>
</table>
GUI  Graphical User Interface
IEC  International Electrotechnical Commission
Java Programming language
JPDA Java Platform Debugger Architecture (provided by JavaSoft)
JSwat Debugger environment (open source, from Blue Marsh Softworks)
LAAS Laboratory for Analysis and Architecture of Systems (CNRS Laboratory)
LISP List Processing (programming language)
MIL-STD Military Standard (Department of Defense, United States of America)
MoD Ministry of Defence (United Kingdom)
NASA National Aeronautics and Space Administration
PASCON Product Assurance and Safety Consortium (ESTEC Project)
PDDL Planning Domain Description Language
RETE Algorithm name (latin for “net”)
RIACS Research Institute for Advanced Computer Science (NASA Laboratory)
RTOS Real Time Operating System
SPAAS Software Product Assurance for Autonomy on-board Spacecraft
SPEC Software Product Evaluation and Certification (ESTEC project)
TC Telecommand
UAV Unmanned Aerial Vehicle
UCAV Unmanned Combat Aerial Vehicle
UGV Unmanned Ground Vehicle
VM Virtual Machine
VxWorks Product (Real Time Operating System) from WindRiver Systems
WITAS Wallenberg laboratory for research on Information Technology and Autonomous Systems
WO Work Order
2 Project Organisation

2.1 Objectives

SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) is an ESA project (contract ESTEC 14898/01/NL/JA). The project objectives were to investigate dedicated software product assurance measures to support autonomous functions both for nominal spacecraft operations and for fault detection, identification and recovery management, i.e., how to ensure safety and dependability of autonomous space software and especially of software in charge of autonomous functions dedicated to the spacecraft safety and dependability management. Special attention was put on software product assurance for advanced autonomy techniques (artificial intelligence, self learning techniques, etc.).

The project was split in two phases. The first phase investigated the lessons learnt from autonomous non-space applications, the software product assurance requirements and then methods, tools and procedures, for autonomous space systems. Special autonomy software safety aspects were then investigated and an implementation plan was elaborated for the second phase. The second phase was dedicated to the definition of software functions (on-board and in the ground system) for the safety of spacecraft with autonomy, and to their implementation and assessment through a pilot application.

2.2 Consortium

The SPAAS project was performed by a consortium led by EADS Astrium SAS (called Astrium SAS when the project started) with Axlog Ingénierie and LAAS-CNRS (Figure 1)

![SPAAS Consortium Diagram](image)

**Figure 1 — SPAAS Consortium**

The consortium relied on the EADS Astrium knowledge and experience in space systems and space software development and validation processes, complemented by Axlog Ingénierie skills in advanced software technologies for large critical software systems, and LAAS-CNRS expertise on dependable computer systems and on autonomous systems (it is worth noting that two research groups from LAAS-CNRS were involved in the project covering both safety and dependability of systems and software, and architecture and software technology for autonomy).

**People involved in SPAAS Project:**

**ESTEC:** Maria Hernek (Technical Officer); Pablo Arrabal (Contract Officer)

**EADS Astrium:** Jean-Paul Blanquart (Project Manager), Jacques de Urtasun (Contract Responsible) with Frédéric Deladérière, Christophe Honvault, Xavier Méchin, Bruno Meijer, and Hervé Schindler.

**Axlog Ingénierie:** Jean-Clair Poncet (Axlog SPAAS Responsible) with Nelly Lécubin, Vincent Legendre, Julien Léonard and Geobert Quash.

**LAAS-CNRS:** Félix Ingrand (LAAS-CNRS SPAAS Responsible) with Sara Fleury, Malik Ghallab (from “Artificial Intelligence and Robotics” Research Group), David Powell and Pascale Thévenod (from “Fault Tolerance and Dependable Computing” Research Group).
2.3 Study Logic

The SPAAS project was split in two phases dedicated respectively to the investigation of the position of the problem and analysis of potential solutions, and to the development and experimentation on generic software components capable to improve dependability and safety on-board autonomous spacecraft.

The first phase (Figure 2) was composed of five work packages, each one dedicated to the elaboration and delivery of a Technical Note.

Figure 2 — SPAAS Phase 1 Study Logic

The first work package investigated the lessons learnt from autonomous non-space applications (Technical Note 1). Based on these lessons learnt:

- The second work package identified the software product assurance requirements applicable to the development and validation of software for autonomy functions on-board space systems (Technical Note 2),

- The third work package identified and discussed methods, tools and procedures to support the requirements for autonomy functions on-board space systems (Technical Note 3).

Based on these intermediate results, the fourth work package elaborated on problems and potential solutions related to advanced software techniques for autonomy and safety (Technical Note 4), and the fifth and last work package in SPAAS Phase 1 proposed an implementation plan (Technical Note 5) for the development and experimentation of SPAAS software components in the second phase.

The Phase 1 started in May 2001, and Final Phase 1 presentation took place in March 2002.
The second phase was dedicated to the definition of software functions (on-board and in the ground system) for the safety of spacecraft with autonomy, and to their implementation and assessment through a pilot application. It was composed (Figure 3) of three logical sets of tasks (arranged in five work packages).

Two parallel sets of tasks were dedicated to the specification, development and validation of two generic software components to support dependability and safety on-board autonomous spacecraft:

- On-board software component (“safety bag”), with specification, development and tests under work package 6 (software data package and associated software documentation package), and metric-based validation (part of Technical Note 7),

- Ground-based software component (“plausibility checker”), with specification, development and tests under work package 7 (software data package and associated software documentation package), and metric-based validation (part of Technical Note 7),

The final set of task corresponded to the instantiation and experimentation of both on-board and ground SPAAS generic components, in a pilot application whose results and lessons learnt were reported and discussed with general project recommendations in Technical Note 6.

The Phase 2 started in September 2002, and SPAAS Final presentation took place in November 2003.
2.4 Project Outputs

The project technical outputs are presented by categories in the following sections and tables. Note that in addition to the project deliverables, three papers were published and presented in international workshop and conferences (§2.4.3, Table 4).

2.4.1 Technical Notes

<table>
<thead>
<tr>
<th>Table 1 — SPAAS Technical Notes</th>
<th>Issue</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>TN1 “Lessons learned from autonomous non-space applications”</td>
<td>1.0</td>
<td>27/7/2001</td>
</tr>
<tr>
<td>TN4 “How to ensure safe decisions in autonomous space systems. Candidate techniques for autonomy on-board spacecraft”</td>
<td>1.0</td>
<td>10/7/2002</td>
</tr>
<tr>
<td>TN5 “Plans for implementing assurance software for autonomy functions”</td>
<td>1.0</td>
<td>26/8/2002</td>
</tr>
<tr>
<td>TN6 “Experimentation of SPAAS reusable components”</td>
<td>1.0</td>
<td>13/2/2004</td>
</tr>
<tr>
<td>TN7 “Identification, collection and assessment of certification requirements”</td>
<td>1.0</td>
<td>22/3/2004</td>
</tr>
</tbody>
</table>

2.4.2 Software

<table>
<thead>
<tr>
<th>Table 2 — SPAAS Software</th>
<th>Issue</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWB Software Data Package: on-board component (safety bag): SCB (code and associated files); SDB (documentation)</td>
<td>1.0</td>
<td>13/2/2004</td>
</tr>
<tr>
<td>SWG Software Data Package: ground component (plausibility checker): SCG (code and associated files); SDG (documentation)</td>
<td>1.0</td>
<td>13/2/2004</td>
</tr>
</tbody>
</table>

2.4.3 Dissemination, presentations, published papers

<table>
<thead>
<tr>
<th>Table 3 — SPAAS Presentations</th>
<th>Issue</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS1 SPAAS Final Phase 1 Presentation, ESTEC, Noordwijk</td>
<td>1.0</td>
<td>25/3/2002</td>
</tr>
<tr>
<td>PS2 SPAAS Final Presentation, ESTEC, Noordwijk</td>
<td>1.0</td>
<td>19/11/2003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4 — Published Papers</th>
</tr>
</thead>
</table>
3 Synthesis of results

3.1 Investigations on software autonomy and dependability

3.1.1 Autonomous non-space systems

The aim of this task, reported in [SPAAS TN1], was to assess the state-of-the-art in software product assurance techniques for critical applications, focussing particularly on autonomous systems.

3.1.1.1 Survey of standards and studies

Major standards relevant to critical systems and software product assurance were analysed. The first two standards give general considerations regarding:

- **Software development and documentation [ST 08]**. Unlike former standards for software development, this standard does not assume a classical “waterfall” development model but allows an open architecture development scheme that can allow multiple program strategies, including evolutionary development. The standard is flexible and may be tailored by the software acquirer to meet specific needs, e.g., those of autonomous systems.

- **Standard practice for system safety [ST 11]**. This standard gives a standard practice for defining the requirements of an acceptable system safety program for Department of Defence acquisitions and provides a common basis for expectations of a properly executed system safety effort. The standard is not specific to software. An interesting feature of the standard is its flexibility in allowing different programs to negotiate different scales of acceptable mishap risk, with decisions being made at the appropriate level of authority.

The next four standards pertain directly to software-based systems used in critical applications.

- **Programmable safety-related systems [ST 09]**. This standard aims to define a generic and application sector independent approach for all safety lifecycle activities of systems using programmable electronic systems. The standard prescribes specific but alternative product assurance techniques, but gives little justification regarding acceptable combinations of techniques. Furthermore, the criteria for compliance are not well defined.

- **Software-based systems for railway applications [ST 12]**. This set of standards broadly follows [ST 09], but specializing it towards railway applications. An interesting aspect is the emphasis on a data lifecycle in addition to the software lifecycle. It also introduces the important concept of a safety case to justify the level of safety achieved by the target system.

- **Safety-related software in defence equipment [ST 06]**. The standard defines the requirements for procedures and practices for developing safety-related software for UK military defence systems. Guidance is given on how to make conformance easier to achieve and assess. Particular emphasis is placed on the use of formal methods.

- **Software in civilian aircraft [ST 07]**. This standard is recognized by all the international certification authorities for approval of software on aircraft. It focuses on the software aspects of system development, and provides guidelines for the production of software for civil airborne systems and equipment in the form of: (1) objectives for software life cycle processes; (2) descriptions of activities and design considerations for achieving those objectives, and (3) descriptions of the evidence that indicates that the objectives have been satisfied. It requires that explicit evidence be provided to demonstrate that an orderly, traceable, repeatable, and rigorous software development process has been followed. The major impact of Do 178B falls into the area of testing and tools.

Our analysis of the four above standards has focussed on the assurance techniques that they recommend, in terms of design strategies for on-line assurance (fault tolerance) and requirements for off-line assurance (verification and validation).
The seventh and final standard is concerned with the general issue of the evaluation of information technology products [ST 10]. This is in fact a series of standards that give methods for measurement, assessment and evaluation of software product quality. They are intended for use by developers, acquirers and independent evaluators, particularly those responsible for software product evaluation. Emphasis is put on the use of validated “metrics” to measure objectively the quality attributes of the software. Here, the term “metric” is defined as a quantitative scale and method that can be used for measurement. These standards describe neither methods for evaluating software production processes nor methods for cost prediction, although software product quality measurements may, of course, be used for both purposes.

Then, two outputs from previous ESA studies were analysed. The first of these, [RD 05], aimed to define for space software a set of software criticality classes similar to those defined in [ST 07] for civil aviation. The second study, [RD 06], was focussed on software product assurance techniques. The study led to a set of documents that are heavily inspired by [ST 10], but with an important and largely successful effort towards conceptual clarification. The notes also include updates to the state-of-the-art and space-specific aspects. The documents give recommendations for techniques and measures that are considered appropriate for space software categorized according to the [RD 05] software criticality classes.

3.1.1.2 Airborne autonomous systems

Recent developments in airborne autonomy were analysed. Major programs are currently led in the military domain, which led to the development of unmanned vehicles for scouting and battle damage assessment purposes. These autonomous techniques are now improved and all major countries own a fleet of unmanned aerial scout vehicles (drones). Further autonomy developments intend to enlarge the scope of potential Unmanned Aerial Vehicles (UAVs) to combat actions (so-called Unmanned Combat Aerial Vehicles, UCAVs) and to increase the size and performance of existing scouting vehicles from the range of a model building like vehicles to the one of current fighter and recognition aircraft. A panel of the projects in progress was presented and a set of recommendations for development of military autonomous systems was proposed, followed by a panorama of different autonomous airborne systems, developed using different techniques for different purposes. This survey also covered the work led in the domain of patrol of UCAV and the emergent technology of micro-drones, which addresses some issues totally opposed to classical drones as it considers the design of very small autonomous or partially autonomous aerial vehicles. The other important aspect of autonomous military airborne systems is the research led to improve classical and cruise missile capacities by embedding decision software. Aspects of software product assurance for military autonomous airborne systems are not well known as they are confidential parts of the projects or, at least, have never been published or communicated. Thus, only assumptions can generally be formulated but it is obvious that military developments for autonomous applications involved compliance to existing norms as for classical military systems.

Concerning recent developments in civilian autonomous airborne systems, two major applications are rising up, the first one for traffic or forest fire control and survey, the other one to improve communication channels, replacing a relay satellite by an high altitude autonomous aircraft. Most of the civilian autonomous aircraft projects are based on vertical take-off and landing aircraft because of their ability to fly stationary or a very low speed. Product assurance in these projects is better known than for military applications but generally relies on the same principles.

3.1.1.3 Waterborne autonomous systems

Autonomous underwater vehicles for civilian applications have been investigated since the 80's but remain research projects, even if commercialisation of such concepts can be envisaged soon. Military applications for autonomous underwater vehicles also exist, but information available on the development of military autonomous submarines is very thin and not really interesting for a software product assurance aspect. Civilian applications of submarine autonomy cover Autonomous Oceanic Sampling Networks that consist of large networks of autonomous submarines for ocean survey and scientific measurement. This kind of architecture introduces automated functions for fleet
management, including mission planning and fault tolerance at a distributed system level. They also cover the development and tests of solar powered autonomous underwater vehicles. Software product assurance is tackled through an incremental simulation approach for testing and validation of developed components and by real testing of final vehicle prototypes. Finally, the survey addressed a European project for development of an autonomous software architecture dedicated to the underwater vehicles, including the software and AI techniques used to address this specific autonomy problem.

3.1.1.4 Terrestrial autonomous systems

An illustrative example of ground autonomous vehicles for military applications is the UGV Demo program which was started a few years ago and is now starting a new phase including a real deployment of autonomous vehicle patrols. The civilian domain was addressed for both road and rail applications, corresponding to current research and already deployed vehicles. The needs for autonomy in such systems are generally satisfied using an automation approach and thus the software product assurance relies on traditional formal and modelling methods for automation. In order to enlarge the scope of ground autonomy, we also addressed the LAAS Architecture for Autonomous Systems and detailed software product assurance issues in complex distributed autonomous ground systems. Two other applications of autonomy were presented in other domains, first for traffic survey and control and second for an autonomous wheelchair concept developed to help disabled people.

3.1.1.5 Product assurance for AI-based autonomous systems

The key AI concepts of knowledge representation and inference are first discussed from the viewpoint of product assurance. Although separation of knowledge representation and inference is definitely advantageous from a product assurance viewpoint (domain experts need only to be concerned by domain-specific knowledge), it is recognized that some intertwining of declarative and procedural knowledge is often unavoidable. Issues of product assurance are then discussed in terms of a high-level classification of AI-based systems according to (a) whether the system function is determined by the designer or, instead, emergent from examples, and (b) whether or not logical consistency is guaranteed. As would be expected, software product assurance is facilitated when a logically consistent system is defined \textit{a priori} by the designer.

Four technical hazards specific to knowledge-based systems are identified:

1. Knowledge base “wrong”: beliefs may be incorrect or data may be missing.
2. Unsound inference: the knowledge base may be correct, but inferences drawn from it may be wrong because the inference procedures being used may be unsound in some way.
3. Unforeseen contingencies: the knowledge base may be correct, but reasoning based on it may break down when it is confronted with some unusual situation not foreseen by the designer.
4. Specificity of decision criteria: the decision criteria built into the system may not be universally acceptable, i.e., they could have adverse side effects in certain situations.

Of these, hazards 3 and 4 are symptomatic of the apparently conflicting requirements that (a) a critical autonomous system be capable of reacting sensibly in situations unanticipated by the system designers, and (b) that its designers can provide assurance that it will do so.

The following main conclusions can be drawn:

- Learning systems are less amenable to dependability and safety arguments than those whose knowledge and inference mechanisms are determined \textit{a priori} by the designer.
- Separate knowledge representation is a key aspect that makes verification and validation of AI-based systems different to that of classical software engineering.
- Only two (complementary) approaches seem feasible for ensuring safe autonomous operation in unanticipated situations:
  - Extensive simulation testing, preferably with an automated oracle.
On-line assurance techniques, such as the safety bag/supervisor approach.

- An evolutionary program development strategy should facilitate a progressive refinement approach in which critical autonomous system capabilities may be addressed first.

### 3.1.2 Requirements for on-board autonomy software

The aim of this task, reported in [SPAAS TN2], was to identify software product assurance requirements applicable to the development and validation of software for autonomous space systems.

The main characteristics of advanced autonomous systems were first described, as a support to the identification of appropriate software product assurance requirements.

The general framework is described through a conceptual model of planning systems. This model was used as a reference for the surveyed architectures. It introduces the notions of state-transition systems, with transitions occurring either on events (uncontrolled or contingent transitions, corresponding to the internal dynamics of the system), or on actions (controlled transitions). Actions are given to the system by a controller according to some plan, synthesised to achieve some goal by the planner. A restricted model is discussed with assumptions on the system (finite, observable, deterministic, static system with restricted goals and implicit time). Then extended models were discussed through relaxing the various assumptions. These assumptions were then discussed in terms of characteristics of advanced autonomous software, and the association of metrics was analysed, highlighting the difficulty of defining general relevant metrics for such systems.

Then a survey of autonomous systems was proposed, focusing on the Deep Space 1 and Remote Agent experiments, Unmanned Aerial Vehicles and Autonomous Mobile Robots.

The Remote Agent experimented in Deep Space 1 is based on a constraint-based planner/scheduler, a reactive executive and a Mode Identification and Recovery system. The planner/scheduler elaborates plans according to the goals, the resource constraints, the expected initial state and the planning horizon. The executive is the central part of the Remote Agent, in charge of translating high-level actions of the plan into a stream of timed low-level commands. It relies on the Mode Identification and Recovery system to support low-level sensor interpretation and commanding, and fault management and recovery.

The case of Unmanned Aerial Vehicles was illustrated through the WITAS autonomous helicopter for road traffic supervision project. It is based on a multi-layered architecture with a deliberative layer (high-level services, planners, recognition packages, etc.), a reactive layer (reactive programs interacting with other layers or parts of the architecture) and a process layer (concurrent computation of feedback control loops). These three layers interact with two information repositories, the knowledge structure repository and the geographic data repository.

Autonomous mobile robots have to interact rationally with a variable and dynamic environment for achieving a wide diversity of tasks. Their architecture has to provide a suitable framework for the interaction between deliberation and action, combining goal-oriented deliberation, time constrained context-dependent decision making, reflex action and second step global correction. Centralised architectures are proposed such as the Task Control Architecture, or hybrid layered architectures such as the “3T” architecture based on a skill manager, a sequencer and a deliberative planner.

After an analysis of the experiences in advanced autonomous systems and particularly in the space autonomous systems, the lessons learned on the software product assurance approaches applicable to the software process on the one hand and on the software product on the other hand were given. These approaches are based on the return of experience from the following projects:

- The successful Deep Space One Probe, with high return of experience communication. The project was oriented by two points of view: the Spacecraft Test people and the Mission Operations people. The project reports focus on the implemented Remote Agent Experiment and its development/operation challenges.
The successful Mars Pathfinder project, the first really highly autonomous spacecraft developed within NASA’s “Faster, Better, Cheaper” paradigm. The project analysis focuses more on the development organisation and approaches.

The advanced autonomous systems and their related development lifecycle and tools. The new NASA choices for autonomous spacecraft abilities were described. Some new areas of interest were also raised, dealing with the validation of data (because of the separation of the programs and the knowledge (data) they use) independently of the programs. The experiences are also enriched by the analysis of a specific RIACS Workshop dealing with this subject.

The lessons learned were then arranged and mapped on the process (the development life cycle (including the operational phase)) and on the product itself. These lessons learned were translated into preliminary recommendations, concentrating mainly on the characteristics of autonomous software. The process breakdown is analogous to the ECSS-40A space software engineering standard, i.e., a requirements phase, a design phase, a verification and validation (qualification) phase, an operations phase and a maintenance phase. Additional requirements deal with critical functions, system interface, in-flight software modification or are more general, taking into account general life cycle needs.

The same approach was used to provide recommendations on the software product.

Then guidelines on the architecture, fault tolerance, verification and validation for autonomous systems were proposed. The synthesis of software architectures for autonomous systems shows a tendency towards layered architectures with a functional layer (built-in action and perception capabilities, processing functions and control loops, interfaced with sensors and actuators through a logical layer), an execution control layer (task control and coordination according to task requirements) and a decision layer (planning and plan execution supervision). The functions are organised in a network of modules. Promising approaches were studied about the definition of generic modules and automatic generation of modules (code generation). Advanced solutions for the execution control layer and decision layer were also discussed, with focus on the criticality of the execution control layer. Fault tolerance mechanisms were discussed, emphasising the interest of the on-line monitoring of safety properties and reactivity with respect to actual availability of resources in real-time. Verification and validation of autonomous systems were finally discussed, identifying needs related to high-level specifications and system design, and to the mapping to low-level languages for automated checking. Some examples of studies were described on the development of analytic verification methods in complement to traditional testing methods.

This task concluded with the ECSS analysis and recommendations. From the characterisation of advanced autonomous, their software product assurance and development approaches and from the identification of guidelines, the ESA ECSS-Q-80 (Space Software Product Assurance) and ECSS-E-40 (Space Software Engineering) were analysed (issues B draft 1). The analysis was performed through the “filter” of the recommendations on the process and the product, recommendations derived from the lessons learned of the actual projects and workshops. Finally, recommendations were given for the applicability of the ECSS to autonomous software development and validation.

### 3.1.3 Methods and techniques for on-board autonomy software

The aim of this task, reported in [SPAAS TN3], was to identify the software product assurance methods applicable to the development and validation of autonomous space systems, and especially those methods appropriate to fulfil the requirements identified in the second task and reported in [SPAAS TN2].

#### 3.1.3.1 Autonomous space systems

We first analysed the context representative of autonomous space systems, based on a general description of the mission, architecture and software product assurance issues in the Rosetta project, completed by a survey of software product assurance related issues in existing autonomy projects, elaborating on the analysis in the first two tasks of SPAAS project reported in [SPAAS TN1; SPAAS TN2].
Rosetta is an interplanetary vehicle with a long and complex mission including interplanetary cruise phases, planet gravity assisted swing-bys, rendezvous with the comet roughly 9 years after launch, and detailed comet study and in situ measurements (with a landed science surface package) during 1½ year. The length and complexity of the mission, the communication delays and available throughput lead to strong requirements for autonomy as well as dependability and safety. The main requirements were discussed, with focus on those related to software fault tolerance, software modifications, ground support, and software design and verification. This synthesis showed that there are general requirements on the high-level autonomy needs, and more specific requirements on functions needed to support autonomy (e.g., support for on-line software modifications, ground controllability). Concerning software, dependability is enforced quite classically on the one hand through a hierarchical fault management and provision for an independent safe mode, and on the other hand through software correctness with a focus on validation. The software and hardware architecture was then described, and the main dependability related features were identified with the list of software product assurance methods applied in the project, and the adopted hierarchical fault management approach.

We then analysed in the context of autonomous space systems, the product assurance approaches used or recommended in major autonomy projects in various industrial or research domains. The lessons learnt from these experiments focus on the importance of a rigorous development and validation approach, strongly linked with the global system process and with explicit consideration of a safety lifecycle. Evolutionary, incremental or spiral software process models are recommended, with early and strong prototyping and simulation support. The focus is also put on the definition of the mission, as well as on the allocation of roles between the autonomous system and the support from human operators. The interest of formal approaches for development and validation, and of automatic generation (of code, of test cases, or test oracles) is also highlighted. Most autonomy projects adopt a layered architecture for software and knowledge representation, with a mapping to the abstract level of information and processing, to the temporal constraints and to the criticality, with a hierarchical fault management approach, with possibly safety supervision approaches. It is generally reported that testing autonomy software is very time-consuming and should be automated and supported by tools as much as possible, as well as it should be complemented with other verification and validation techniques such as prototyping, simulation, formal proofs, etc.

3.1.3.2 Software product assurance and methods

We proposed a synthesis and classification of the software product assurance methods and procedures from the analysis provided in the survey performed in the first task [SPAAS TN1], of the major standards and norms and the former ESA project PASCON WO12.

The recommendations were identified, ranging from the least means-prescriptive standards (MIL-STD-498, MIL-STD-882D, MoD 00-55, Do 178B/ED 12B, providing general goal-oriented recommendations and guidance), up to IEC 61508 and the derived CENELEC EN 50126/8/9 standards providing explicit lists of recommended (or not recommended) techniques and methods for software architecture design, strategies for fault tolerance and software verification and validation. This list was completed with the survey of software dependability and safety techniques performed in the PASCON WO12 project, covering design and coding practices, process practices, static methods and dynamic methods. All these methods and techniques were provided with their recommendation level according to the software criticality category and with their applicability within the lifecycle, as indicated in the corresponding source documents.

We then proposed a synthesis of the identified software assurance methods, classified according to the four basic means for dependability: fault prevention, fault tolerance, fault forecasting and fault removal. Fault prevention gathers methods related to the general organisation, system and software process, support processes (quality assurance, documentation, configuration management, etc.), design methods and design analysis methods, including formal approaches, design and programming constraints, reviews, simulations, prototyping, etc. Fault tolerance gathers methods dedicated to the detection of errors (through data flow or control flow monitoring: defensive programming, watchdogs, etc.), and methods aiming at either preventing error propagation (segregation/partitioning, safety
supervision, wrapping, etc.), or at providing service continuity through diversity (diversity of design: N-Version Programming, recovery blocks, etc.; diversity of execution with temporal redundancy). Fault forecasting gathers techniques for analysing the propagation of faults (failure modes, effects and criticality analysis, fault tree analysis, etc.), techniques for quantitative reliability estimation, fault injection techniques (when used for the experimental analysis of failure modes or estimation of dependability parameters such as error latency), and specific techniques such as common mode analysis or hardware/software interaction analysis. Finally fault removal gathers the techniques to support the identification of software faults to correct, including static analysis techniques (code inspection, walkthroughs, reviews, code analysis), and dynamic techniques i.e., testing with the various techniques to generate test cases and assess the test coverage.

3.1.3.3 Product assurance software for autonomous space systems

We analysed the characteristics of software for autonomous space systems, and the applicability of the identified software product assurance methods to software for autonomous space systems. The main characteristics of space autonomy software come from the constraints and needs induced by space autonomy on the software functions (number of functions, complexity, maintainability, temporal constraints and criticality). The impact of these constraints on the software characteristics was discussed, highlighting the technological issues (model-based, rule-based approaches, etc.), the architectural issues (layered architecture, separation between knowledge representation and procedures manipulating the knowledge, organisation of fault management), and the software characteristics in terms of complexity, criticality, controllability, observability, predictability, etc. The impact of autonomy on spacecraft dependability and safety was analysed and discussed, based on analytical modelling and evaluation, showing that the improvement of dependability (decrease of down-time and of mission unreliability) and of safety (increase of the mean time to catastrophic failure) are of the same order of magnitude as an increase of the proportion of events handled autonomously. However the analysis also highlighted the importance of the on-board and ground support to the dependability of the autonomous spacecraft, the coverage of the provided mechanisms appearing as at least as important for the system properties as the degree of autonomy.

The applicability and the interest of the identified software product assurance methods for space autonomy software, according to their characteristics and to the lessons learnt on product assurance in autonomy project, were then analysed.

It finally appears that though most product assurance methods can be applied to autonomy software, or at least to some parts of the software of an autonomous system, some may however decrease the advantages that could be expected from some autonomy technologies (e.g., design or programming constraints), while some other (e.g., testing) may become very time-consuming and may not bring in as much confidence as for less complex software.

Confirming the lessons learned from autonomy projects as reported in [SPAAS TN1; SPAAS TN2], it can be recommended to adopt an incremental development approach integrated within the global system and safety lifecycle, in association with strong and early prototyping and simulation support. Benefit can be drawn from the natural layered and modular approach of autonomy software, as well as from their formal nature to support the development and validation with formal approaches, both for the procedures and for the data and knowledge representation, the validation of which is particularly important.

The process must of course cover the development and validation of further modifications for corrections or evolutions, as well as the initial design for maintainability of software. The hierarchical fault management concept can usefully be exploited to support a safety management approach with on-line verification and enforcement of safety properties, combining mechanisms at various levels from global top-level safety properties down to more specific ones.
3.1.4 Advanced software technologies for autonomy and safety

The aim of this task, reported in [SPAAS TN4], was to study techniques that are suitable for future space on-board autonomous applications, focusing on advanced software technologies for autonomy.

3.1.4.1 Space autonomy functions

We first identified the principal space autonomy functions that could be envisaged in the short term, i.e., applications for planning technology. Several planning techniques have been developed, all concerning the possible ways to determine sequences of actions that shall be executed by one or several entities in a given environment. We first identified and described the planning goals and general techniques, and detailed the PDDL language, which has been developed for autonomy purposes, giving a good idea of planning requirements and possibilities. The scope of applicable planning techniques and languages was then enlarged, introducing PRODIGY and GRAPHPLAN, heuristic based planning and learning methods. The planning technologies may be associated to planning algorithms in order to improve plan computation and adaptation to new situations. Some of these aspects were illustrated through constraint programming and genetic algorithm approaches, and the adequacy of these planning techniques to the space requirements was discussed. The space requirements illustrated here resulted from the second task of the project [SPAAS TN2]. The presentation of planning techniques has been supported by some examples of applications, detailing some product assurance aspects. Finally the potential applications of these techniques to diagnosis and fault detection were discussed.

3.1.4.2 Knowledge representation

Two main techniques were addressed: Rule Based Reasoning and Case Based Reasoning.

Rule Based Reasoning: the first order logic theory is commonly used to represent rules and models rule inference. The main inference principles were analysed, and described in [SPAAS TN4] using examples. A detailed description of the most used algorithms like the RETE one was provided, including the description of forward chaining, which is the basis of RETE, of backward chaining, of the table-based and tree-based knowledge indexing mechanisms.

Learning is an important aspect of rule-based systems. Learning may be used as a tool to learn new rules or new ways to trigger rules in the inference process, but learning algorithms could also be implemented using rule-based systems.

We then surveyed languages that have been used to design rule-based systems. This could be some language like LISP, which was one of the first to be used or other specifically designed languages like CLISP. Adequacy of Rule Based systems to space requirements was analysed, as well as the existing applications of these systems to autonomous vehicles and their associated product assurance approach.

Like Rule Based systems, Case Based Reasoning systems are knowledge-based systems that use a specific form of knowledge. In rule-based systems, knowledge corresponds to a set of rules that can be activated and inferred to give a solution. In Case Based Reasoning systems, knowledge corresponds to recorded problem situations and solutions that are used as reference to infer a solution from the current situation.

Using Case Based Reasoning necessitates dealing with different notions relative to case abstraction, solution extraction, learning, etc., leading to various problems encountered and possible solutions when considering case abstraction, case retrieval in the case base, case comparison, case reuse, and case adaptation when no similarity can be highlighted. Case refinement and coherence checking through case verification are also tackled. We finally analysed how learning can be used in order to enrich the case base, storing new cases to enlarge the system's potential, discussed the adequacy of Case Based Reasoning systems to space requirements, and described some applications of this technique to autonomy and failure management.
3.1.4.3 Techniques for autonomous component implementation

**Constraint Programming**: Constraint Programming is a numerical technique used to solve huge combinatorial problems in integer and real number expression. It has been widely used in industry for different purposes like crew organisation, workshop scheduling, and other process optimisation. It has been used for autonomy purposes for several years, mainly for planning and scheduling. Constraint Programming was described and illustrated in [SPAAS TN4] through the so-called N-queens classical combinatorial problem example. This sample problem was first introduced to illustrate constraint modelling, before giving a more formal definition of Constraint Programming, introducing constraint satisfaction and the mathematical basis of Constraint Programming and Constraint Logical Programming.

Once these formal structures have been given, we detailed the basic algorithms used to solve algebraically Constraint Satisfaction Problems and introduced a way to build learning algorithms for the Constraint Satisfaction Problem. The choice of this technique has been validated through the analysis of the adequacy of Constraint Programming technology to space requirements. Finally, we presented a set of existing applications of Constraint Programming to autonomy and to space autonomy, including a description of the product assurance approach that has been chosen.

**Genetic Algorithms**: Genetic Algorithms have not been widely used for autonomy purposes. However, some applications exist and they are interesting to implement part of an autonomous behaviour or to perform fault detection, fault recovery and autonomous function verification. These algorithms can be associated to an evolutionary process; they necessitate the definition of specific selection functions and the tuning of parameters.

The major field of application of Genetic Algorithms concerns the optimisation problems, in which a given criterion must be maximised or minimised. The side effect of such algorithms that can lead to premature convergence giving a non-optimal solution or complete inefficiency according to problem nature and structure was analysed and discussed. Genetic algorithms may also be used to implement learning processes, where learning can be expressed as an optimisation process and tackled using a Genetic Programming approach. The adequacy of Genetic Algorithms to major space requirements has been discussed, as well as potential applications of Genetic Algorithms to autonomy, in particular to planning and scheduling tasks in order to implement autonomous agent behaviour, and for testing autonomous software.

**Fuzzy Logic**: this technique is based on Fuzzy Set theory and Possibility Theory, built on Fuzzy Logic and concepts including the linguistic variable notion, fuzzy membership function and the inference mechanism that enables “reasoning” using fuzzy rules and linguistic variables. The mechanism of Fuzzy inference can be applied to a set of rules in order to build a decision module that can be used, for instance, to design a Fuzzy Controller. The association of learning with fuzzy rule based systems enables dynamic update of the set of fuzzy rules by learning new ones. We analysed and discussed the adequacy to space requirements of Fuzzy Rule Based systems and their possible applications to autonomy applications, including Fuzzy Controllers, and the associated product assurance approaches.

**Artificial Neural Networks**: Artificial Neural Networks are a well-known technique that has already been widely used for autonomy purposes but also for many automatic applications. Artificial Neural Networks are based on the observation of the physical structure of human brain. They aim to model the processing of such a brain, in a much less powerful but quite effective way. We analysed the mechanisms of Neural Computing, one of the most commonly used Artificial Neural Network structures, the Multi Layer Perceptron and its associated learning algorithms, another powerful network structure, the Kohonen map network, and finally the recurrent network concept that is a promising technique for future autonomy applications. We analysed and discussed the adequacy to space requirements of Neural Networks and their possible applications to autonomy applications, including Neural Controllers, and the associated product assurance approaches.

**Stochastic techniques**: Two techniques were addressed: Probabilistic Networks and Markov Decision Processes. Probabilistic Networks propagate probabilities from one node to another until reaching a stable state. Probabilities are propagated using Bayes’ Rule on conditional probabilities and algorithms
for belief propagation inside the network since, in Bayesian Networks, beliefs are the states of the nodes. According to some problem structures that involve a large part of uncertainty, it could be interesting to be able to learn a network that matches the problem description and gives a solution according to a given instance. Some methods exist to learn a Bayesian Network. We analysed and discussed the adequacy to space requirements of Probabilistic Networks (taking into account the non determinism aspect of such a decision procedure and the difficulty to acquire good probability distributions according to the knowledge of a given problem) and their possible applications to autonomy applications, and the associated product assurance approaches.

The Markov Decision Process technique is a stochastic approach to the decision problem, enabling uncertain or unknown situation data to be accommodated. The Markov Decision Process technique is based on the Markov Chain Theory, relying thus on a strong mathematical basis. According to the nature of knowledge data (completely known or only partially known), the Markov Decision Process could be Fully Observable or Partially Observable. Both cases involve learning techniques to automatically acquire or refine a Markov Decision Process. We analysed and discussed the adequacy to space requirements of Markov Decision Process and their possible applications to autonomy applications, and the associated product assurance approaches.

On-Board Control Procedures: This survey and analysis of software techniques for autonomy was concluded with a brief description of On-Board Control Procedure technique that has already been selected in space projects and in particular the Rosetta spacecraft.

3.1.4.4 Agents for autonomous systems

Many design choices could be made while designing the software architecture of autonomous systems, the main ones being based on the concepts of Agents and Multi Agent Systems. An agent is a generic concept often used when considering autonomous systems as it enables to describe a system as an autonomous entity through its behaviour. Agent technology and types are generally classified as reactive, cognitive and social agents. Multi Agent Systems extend the concept of agents through the consideration of distribution concepts and involved policies. Multi Agent System concepts were expressed in the study in the most possible generic way, as numerous approaches to modelling of agent systems exist without emergent norms or standards.

Different possibilities exist to organise an agent system, the main ones being the centralised and distributed organisations. We also addressed the different techniques used to allow agents communicate with one another, improving their capacities through knowledge exchange. The use of learning algorithms for agent adaptation and knowledge update is an important issue Generally speaking, learning is an important field of autonomy-oriented techniques as it could enable both adaptation of autonomous software to unpredicted events, and improvement of software response quality. Numerous tools and frameworks have been developed to facilitate the development of agents and agent communities. We identified and discussed the ones that seem to be the most suitable for space autonomy purposes.

The adequacy of agent systems to space requirements was analysed and discussed in the same way as for Constraint Programming techniques, and illustrated different examples of agent applications in the autonomy domain.

3.1.4.5 Synthesis

Advanced software technologies for autonomy finally provide promising ways to address the needs for autonomy of on-board space systems, though the maturity of the associated applicable software product approaches generally still calls for major attention and complementary or alternative techniques especially for safety critical or safety related systems. Conversely (or complementarily) it is worth noting that these advanced software technologies for autonomy were successfully applied as a support to software product assurance, e.g., Constraint Programming for functional testing, and Genetic Algorithms for complex software testing (both examples analysed during the project and reported in [SPAAS TN4]).
3.1.5 Plans for autonomy and safety software components

Based on the four first investigation and analysis tasks, the last task of the first phase of the SPAAS project proposed plans for the implementation, during phase 2, of software components, capable of providing solutions to the need to cope on-line with potential abnormal behaviours or unexpected situations (leading to fault tolerance mechanisms such as a safety supervisor (safety bag) dedicated to the monitoring and preservation of safety properties) and the need to complete traditional verification and validation activities with intensive simulation based on scenarios, in particular for the ground-based plausibility check of an important support to the spacecraft autonomy: the on-board procedures.

These two components are illustrated within the global space system in Figure 4.

![Figure 4 — Safety software components in autonomous spacecraft system](image)

This preparation task to the second phase of SPAAS defined the software development plan with focus on the quality assurance plan and the validation plan following the recommendations from the previous tasks.

These two components were designed and validated as generic components intended for reuse in future autonomous space systems as support to safety. The second phase of SPAAS included an experimentation covering the instantiation process and behaviour validation in a representative space environment, and metric-based assessment of the generic components through the definition of an appropriate subset of properties and metrics analysed and discussed in this fifth task, from the quality model proposed in the ESTEC SPEC project [RD 04].
3.2 Software components for safe autonomous on-board systems

3.2.1 Safety monitoring: the “safety bag” (on-board)

An on-board generic component (the “safety bag”, Figure 5) has been developed to monitor on-line a set of safety properties so as to authorise or not the execution of commands to the spacecraft elaborated by the autonomous software applications.

![Diagram of the safety bag architecture and situation](image)

(DHS: Data Handling System; RTOS: Real-Time Operating System; TC: Telecommand)

**Figure 5 — Safety bag architecture and situation**

The safety bag was developed as a generic component and its experimentation has been performed through a three-month pilot application on hardware, software and safety properties from real space projects.

The safety bag was developed in C and experimented on a real data handling system running both in a Sun/Solaris and in an ERC32/VxWorks environment. The experimentation addressed:

- The evaluation and assessment of real-time performance and safety-related performance: coverage, latency, false alarm rate;
- The investigation of potential improvements or alternative solutions, particularly for the integration of the safety bag within the on-board platform architecture;
- The analysis of safety properties with the aim:
  - To provide methodological support and practical guidance for the definition of relevant safety properties to projects where the safety bag is instantiated and implemented;
  - To assess the capability of the safety bag to monitor efficiently, through reliable information available on-board, the various kinds of safety properties relevant for the different nature of space systems and missions.
Two functions extracted from real space applications have been used to check the behaviour of the Safety Bag component. The first function is an agility function commanding manoeuvres of actuators to carry out attitude control by means of direct telecommands. The second function is an autonomous application generating activity plans by means of time-tagged telecommands.

The principle of the Safety Bag is illustrated in Figure 6. Each time a telecommand is sent to the TC Services to be routed to its final destination, it is intercepted. A transition function evaluates the effect of the execution on the current system state. A new (virtual) state is generated and a Verification function checks whether the safety properties are respected. If the properties are respected, the telecommand is sent to its final destination for actual execution, otherwise it is rejected.

![Figure 6 — Safety bag principle (1)](image)

Because some telecommands may be executed in more than one step (e.g., arm and fire or switch-on and switch-off telecommands), the transition function must have knowledge of pending time-tagged telecommands. This allows, for example, an estimation of the power consumption of an equipment item during its activity. For this reason, the Safety Bag is placed at the interfaces of the standard telecommand services of the data-handling services (Figure 7).

![Figure 7 — Safety bag principle (2)](image)
The Safety Bag intercepts telecommands as well as cancellation commands transmitted to the TC Services by any source. In a first step, the Safety Bag processes the time-tagged telecommands by memorizing them, and the cancellation commands to remove them from the memorized list (if the cancellation does not violate safety properties). Telecommands are then transmitted to the standard TC Services that will process and route the telecommands immediately or at a predefined time. When a telecommand is routed to its destination, it is intercepted by the Safety Bag to verify the safety properties (second step). The verification is then based on the last known state of the system.

In the frame of the experimentation, the safety properties checked by the Safety Bag concerned the values of parameters sent to software applications and the available resources of the satellite.

The Safety Bag has been developed as a generic component and its instantiation for a particular project requires:

- Definition of the content of System State (and when needed the functions that build it).
- Definition of the format of the telecommands processed by the system and in particular the localization of the source and destination identifiers, the size of the data and the potential time-tag.
- Develop the Transition functions. A transition function must be attached to each telecommand that has to be checked. A simple transition function only carries out a raw evaluation of the future System State generally based on worst cases. However, this can lead to the generation of false alarms. A complex transition function implements a model of the system. The estimation is more accurate but requires more processing power and time. Moreover, a complex transition function could be error-prone.
- Develop the Verification functions. One or several verification functions can be developed in order to minimize the number of checks to perform on the foreseen system state.

The definition of the system state content and of the format of the telecommands is done by means of text files used to automatically generate the code that is later compiled and linked with the generic Safety Bag library. This method is simple and allows the optimization of the code running on the final target by avoiding for instance the implementation of an on-board interpreter. For the experimentation, only a few variables have been defined, allowing the main parameters of the spacecraft to be known: position, velocity and time vectors, and available power.

Several transition functions have been developed to estimate the effect of a manoeuvre or an activity on the system including the two-step commands. For example, an image acquisition activity includes the switch-on and switch-off of the optical instrument. The transition function is thus able to evaluate the power consumption of the instrument during its period of activity.

The verification functions developed in the frame of the experimentation checked a subset of the system state generated by the transition function. This subset depends on each telecommand so as to minimize the processing time.

Once the Safety Bag is included within the system, it can be activated and configured by specific telecommands. The configuration defines for instance which telecommand emitters must be monitored. The configuration can be modified on-line (by ground telecommands so as to avoid uncontrolled erroneous modifications of the behaviour of the available safety mechanisms).

During all experiments, the Safety Bag demonstrated a correct functional behaviour. All the telecommands suspected to be dangerous for the system were rejected and only those telecommands. The Safety Bag correctly managed the time-tagged telecommands.

The Safety Bag has been tested in a representative on-board software architecture based on the DHS32 running either on top of a Sun/Solaris or an ERC32/VxWorks environment. No functional difference has been detected between the two environments.
3.2.2 Interpreted procedures validation: the “plausibility checker” (ground)

A ground-based generic component (the “Plausibility Checker”, Figure 8) has been developed to support and complement the ground validation of autonomy software, and especially the on-board control procedures before upload and actual execution.

![Diagram of Plausibility Checker architecture and situation](image)

(DHS: Data Handling System; TC: Telecommand)

*Figure 8 — Plausibility Checker architecture and situation*

The Plausibility Checker was developed as a generic component and its experimentation has been performed through a three-month pilot application on hardware, software and safety properties from real space projects.

The Plausibility Checker was developed in Java and has been experimented in several environments including a standalone host workstation or personal computer, and a workstation connected to an existing facility for validating on-board control procedures. The experiment focused more precisely on the extent, scope and nature of the properties that can be checked through this approach, to provide a useful complement to existing validation procedures. Another aim was to analyse and identify the best approach for such a component, from the definition of reusable specifications (and possibly some support components and generation tools) for the development of project-specific validation benches, up to the development of a fully reusable component to plug into several different project-specific validation benches.

The checking rules allow the definition of the elements that must be checked in order to declare a test successful or not. These rules can apply to Datapool variables (value limits, access checks, etc.) and to TC Services used (e.g., generated telecommands). Some rules can also be defined on the System State. The Plausibility Checker supports the execution of these rules. The definition of the supported rules must be sufficiently generic to support different kinds of missions. In a future version of the Plausibility Checker, the checking rules could evolve to a full test language.
The rules that apply on the System State variables allow the Plausibility Checker to:

- Check if the variable has been modified since the last application of the rules.
- Test the value of the variables against predefined values or ranges of values.

The rules that apply on the Datapool variables allow the Plausibility Checker to:

- Check if the variable has been modified since the last application of the rules.
- Test the value of the variables against predefined values or ranges of values.
- Check if the Interpreted Procedure has the authorization to access the Datapool variable (both read and write access can be verified).

The rules that apply on the telecommand execution allow the Plausibility Checker to:

- Check if the Interpreted Procedure has the authorization to execute the telecommand.

Moreover, the order and time of some events (Datapool variable access or telecommand emission) can be important for the safety of the System. The checking rules allow the Plausibility Checker to verify if the order of specific events is respected, and the current time of the simulation to be checked.

An integrated architecture was developed to facilitate the experimentation of the plausibility checker as well as its future instantiation in space systems (Figure 9). This integrated architecture allows use of a generic Java test environment that provides standard functionalities to manage execution of interpreted procedures in debug mode. ‘Step by Step’, ‘Run to’, ‘Breakpoints’, ‘Watch’ etc. functions are directly provided by the environment through a direct standard interface with the Interpreter.

The Plausibility Checker functions are implemented by registering values to monitor (including Events and TC, CheckRequest), which are added as watch variables inside the environment. At each run step the environment sends updated values to the Plausibility Checker that performs evaluation. For the rest, the graphical user interface is updated to manage the Plausibility Checker. This standard mechanism allows the application to be updated with dynamic monitoring of the Interpreted Application. This architecture is based on the JSwat environment (licensed under the GNU General Public License and freely available in both binary and source code form), written to use the Java Platform Debugger Architecture (JPDA) library provided by JavaSoft.

The following two figures (Figure 10 and Figure 11) show the GUI provided by JSwat environment that have been updated to manage the Plausibility Checker.
For each monitored data, the PC returns the result of evaluation. List of Monitored data, with values displayed in real-time. Predefined data include TC, Event and Request.

Specific Menu added to JSwat to load States descriptions and Rules, Start/Stop the Plausibility Checker. JSwat allowing debug, step by step execution etc. Breakpoints may be added/removed dynamically.

Thumb allows selection of Threads and Procedures to monitor, local variables, and watches to be added. JSwat allows fine VM control, and provides standard high quality GUI.

Other thumb gives access to Messages, Outputs, PChecker results and manages breakpoints. It is possible to monitor the Interpreter stack and parse methods.

Figure 10 — Plausibility Checker environment based on JSwat (1/2)

Figure 11 — Plausibility Checker environment based on JSwat (2/2)
3.3 Synthesis and recommendations

The SPAAS project addressed the software dependability and safety issues for autonomous spacecraft, with a focus on software product assurance approaches and dependability mechanisms applicable to autonomy software.

The survey of software safety and dependability methods, standards and industrial practice highlighted the needs both to complement the verification of autonomy software through intensive simulation and assessment of plausibility properties, and to monitor on-line at least the most important safety-related spacecraft properties. This led to the definition, development, validation and experimentation of generic software components to support dependability and safety of autonomous spacecraft: an on-board safety-bag and a ground-based autonomous procedure plausibility checker, to be used in future autonomous space projects.

The major part of the work needed for the insertion of the Safety Bag in an existing system consists in the identification of the required System State variables and the coding of corresponding elaboration functions as well as of the Transition and Verification functions.

This highlights the importance of the sound identification of the safety properties that must be checked by the Safety Bag. This could not be as generic as the developed safety components. However, starting from this study, further work has been engaged to clarify and make more systematic and sound the process of elicitation, refinement and allocation of dependability and safety properties in complex critical autonomous systems.

In a real on-board space system, due to the limitation in power processing and in memory, the number of critical telecommands to check should be certainly limited as well as the complexity of the functions. It is thus very important to identify the main cause of potential faults in the system. With sufficient processing and memory capacity, the transition function could include an accurate model-based representation of the system.

The plausibility checker does not have the same real-time constraints as the safety bag. However, it has to interact with a large number of elements to control and check the execution of the interpreted procedures. This may imply a great complexity in the use of the plausibility checker. A first simplification has been brought by the integration of the plausibility checker in a generic and standard development environment. All the tests executed with the plausibility checker allowed us to detect the violation of rules and properties. As a result of the study, the existing EADS Astrium product in charge of the interpreted procedures verification could reuse some plausibility checker requirements.

From experimentation of both on-board and space components, it appears that the most important work to perform is to clearly identify the properties and rules the system must respect. The identification of these rules and properties may be difficult in the context of complex autonomous systems. These properties and rules must then be transferred in the transition and verification functions for the safety bag and in the form that can be handled by the plausibility checker. The creation of a methodology covering the identification and the way to handle the safety properties should allow a formalisation supporting the correct utilisation of the components. It is worth mentioning that an internal LAAS-CNRS multi-group project (involving the two research groups participating in SPAAS, “Fault Tolerance and Dependable Computing” and “Robotics and Artificial Intelligence”) on “Autonomous Safety Critical Systems” has been recently launched to address these issues (with contribution of EADS Astrium to the Advisory Board of the project) (http://www.laas.fr/SAC/).

The proposed concepts and a large part of the solutions down to the component level could be fruitfully extended towards embedded real-time software systems in other domains. Though developed and experimented in the context of space systems, resulting in some specific implementation characteristics, many similarities were found and common issues addressed.