Software Safety Supervision On-board Autonomous Spacecraft

J P Blanquart, S Fleury, M Hernek, C Honvault, Félix Ingrand, J. Poncet, D Powell, N Strady-Lécubin, P Thévenod

Session 4A: Fault Tolerance

Software Safety Supervision On-board Autonomous Spacecraft

JP. Blanquart[1], S. Fleury[2]; M. Hernek[3]; C. Honvault[1]; F. Ingrand[2];
JC. Poncet[4]; D. Powell[2]; N. Strady-Lécubin[4]; P. Thévenod[2]

(1) EADS ASTRIUM, 31 rue des cosmonautes, F-31402 Toulouse Cedex 4, France
jean-paul.blanquart@astrium.eads.net (contact author)

(2) LAAS-CNRS, 7 avenue du colonel roche, F-31077 Toulouse Cedex 4, France

(3) ESTEC, Keplerlaan 1, PO Box 299, 2200 AG Noordwijk ZH, The Netherlands

(4) AXLOG Ingénierie, 19-21 rue du 8 mai 1945, F-94110 Arcueil, France

Abstract

This paper presents a study on software product assurance measures and dependability techniques to support autonomous functions on-board spacecraft. An analysis of current standards and techniques in space and other domains, and a survey of software autonomy projects from the point of view of product assurance, dependability and safety are presented. Product assurance measures are proposed, and the paper concludes with the description of two generic software components that have been developed and experimented to provide additional safety mechanisms in autonomous space systems: a "safety bag" in charge of monitoring on-board a set of safety properties, and a "plausibility checker" complementing on ground the validation means for interpreted procedures before they are uploaded and executed on-board.

1 Introduction

Increased autonomy is an important trend in space systems, taking advantage of the increase of the on-board processing power to enable new or more efficient complex missions. This is particularly useful when the ground cannot react in real-time due to communication delays, non-visibility periods, complexity or variability of the context. This raises new challenges for mission reliability and safety, due to context unpredictability, and the criticality and complexity of the software components implementing autonomy functions. Criticality implies to strong software dependability and safety requirements, while complexity makes it more difficult to fulfil such requirements. These peculiarities imply especially adequate software product assurance methodology and software dependability techniques.

SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) is an ESA project (contract ESTEC 14898/01/NL/JA), granted to a consortium led by EADS Astrium with Axlog Ingénierie and LAAS-CNRS [1]. The objectives of the project 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. In other words, how to ensure safety and dependability of autonomous spacecraft software and especially of software in charge of autonomy functions dedicated to

summary previous next

spacecraft safety and dependability management. Special attention was directed towards 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 proposed 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 Standards and Practices

This section analyses the various methods for software dependability and safety, as recommended in standards and norms, or used in industrial practice. Seven standards and norms were analysed:

- US Department of Standards MIL-STD-498 and 882D,
- IEC 61508 standard on programmable safety-related systems,
- CENELEC EN 50126/8/9 series of standards for railway applications,
- UK Ministry of Defence MoD 00-55/6 standards for safety-related software,
- Civilian aircraft DO 178B/ED12B standard,
- IEC 14598 standard on the evaluation of information technology products.

In addition, industrial practices were analysed, from former ESA studies on software dependability and safety (PASCON WO12) and from advanced autonomy projects for airborne, waterborne and terrestrial systems.

It appears that most safety-related software standards pay little explicit attention to autonomy and to the particular advanced software technologies for achieving system autonomy. In practice the recommended set of techniques and methods for safety-related software may not be easily applicable considering, e.g., the size and complexity of the software and of the input and state domains, the dependency of the software behaviour on knowledge bases, etc. [2] This is confirmed by the available reports and studies on advanced autonomy systems, as discussed for instance in a recent specific workshop that addressed the verification and validation of autonomous and adaptive systems [3]. 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 *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.

summary previous next

  - 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 Software for Autonomy

Various software autonomy techniques are available such as rule-based systems, case-based reasoning, constraint programming, genetic algorithms, fuzzy logic, artificial neural networks, probabilistic networks, Markov decision processes, agent and multi-agent systems. A survey was performed, analysing each technique according to its mathematical and algorithmic definition, impact on space architecture and functions, and applicability of current software product assurance standards. A focus was put on issues of interest for autonomy in space systems:

- From a functional viewpoint, on planning and scheduling, diagnosis, and on the notion of on-board control procedures;
- From a product assurance viewpoint, on the applicability of software dependability methods and of the clauses of the software product assurance standards for space systems (European Cooperation for Space Standardization, ECSS [4]).

Usual software design approaches cannot tackle all the difficulties raised by autonomous systems. Because of the complexity and the critical nature of those systems, product assurance is very central. However, product assurance calls for deterministic behaviour whereas autonomy requires the ability to handle nominal and non-nominal situations and events in a wide range of contexts and missions. The combination of all possible states and events excludes an exhaustive representation of those states and transitions in order to prove *a priori* that the autonomy software will always behave as expected. On the contrary, the system must be endowed with some decision capacities on-board that will be able to analyse missions on-line according to the current context (i.e., the current state of the system and its environment) and to decide dynamically on the actions required to achieve the missions' objectives.

The answers are strongly related to software product assurance and they call for:

- A well-defined software architecture that can integrate both strong real-time functions and robust decision capacities. Every part of the architecture must be precisely defined, including its functions, interfaces, inputs and outputs, required temporal properties, limitations, etc, and the overall logical and temporal articulations between these components (see Figure 1).
- Standard components and interfaces to permit coherent and incremental integration of complex and heterogeneous functions.
- As far as possible, automatic code synthesis, the only way guarantee the conformity of the implementation with respect to the design documentation.
- Specific tools to check dynamically the consistency of the system.
- Specific tools to design the two main functions of the decision level: the planning and the supervision of the tasks or actions.

summary previous next

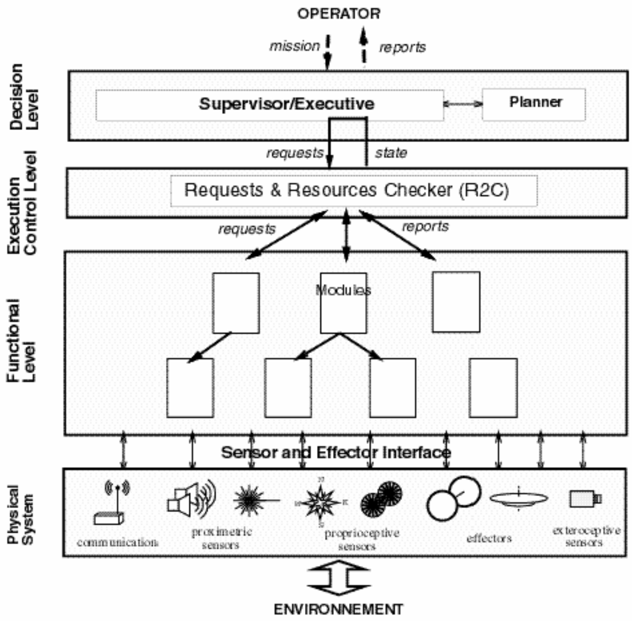


Figure 1: The LAAS three-level hybrid architecture [5]

4 Autonomy Software Dependability and Safety

It finally appears that autonomous systems and especially those based on advanced autonomy technologies and artificial intelligence (AI) pose some significant challenges regarding software product assurance. They are a relatively new trend in real-world critical embedded applications, particularly in space systems, and there have been few studies aimed specifically at defining appropriate assurance techniques. However, several tentative conclusions may be drawn [6]:

- The problem of verifying and validating knowledge-independent components of an AI-based system (e.g., inference mechanisms) is similar to that of classical software engineering.
- Separate knowledge representation is one key aspect that makes verification and validation of AI-based systems different to that of classical software engineering. Checking the consistency and completeness of the knowledge representation has thus received deserved attention. Several authors however underline the advantages, from a product assurance viewpoint, of having domain-specific knowledge represented separately from procedural mechanisms making use of it, since domain experts may more readily check it. Moreover, logic-based inference mechanisms may allow formal proof of correctness properties.
- Learning systems, whose function emerges from training examples or during operation, prove to be quite robust in practice. Nevertheless, they are less amenable to dependability and safety arguments than those whose knowledge and inference mechanisms are determined *a priori* by the designer.

summary previous next

- Although autonomous systems are required to operate for extensive periods of time without human intervention, it is important that autonomous systems also support human intervention when necessary. However, when humans and AI-based systems are to interact synergistically, new human factor risks may be introduced.
- Autonomous operation can significantly impact software development in that domain-specific knowledge needs to be encoded early on. An evolutionary program development strategy should facilitate a progressive refinement approach in which critical autonomous system capabilities may be addressed first.
- The most significant challenge in the use of AI-based techniques for autonomy is that of unanticipated and complex situations in which the system is nevertheless expected to act sensibly. As mentioned in Section 2, there are only two apparent (complementary) ways to address this challenge:
    - Use extensive simulation testing to increase statistical confidence that the autonomous system will behave as expected. For really extensive simulation testing, some form of automated oracle should be envisaged. For space systems, this does not only concern the autonomous on-board applications, but also the procedures loaded or uploaded to be interpreted on-board ("on-board control procedures").
    - Use on-line assurance techniques, such as the safety-bag or safety supervisor approach to ensure that catastrophic failures are avoided, which implies some form of graceful degradation [7]. The generalization of the safety bag concept towards "active safety management" is also an interesting direction for future research [2].

In addition to recommendations on design, validation and product assurance techniques, there is thus a strong need for "functional assurance software components", on the one hand to support complementary validation through extensive simulation testing, and on the other hand to provide safety-oriented monitoring and protection on-board during the operation phase.

5 Components for Safe Autonomous Spacecraft

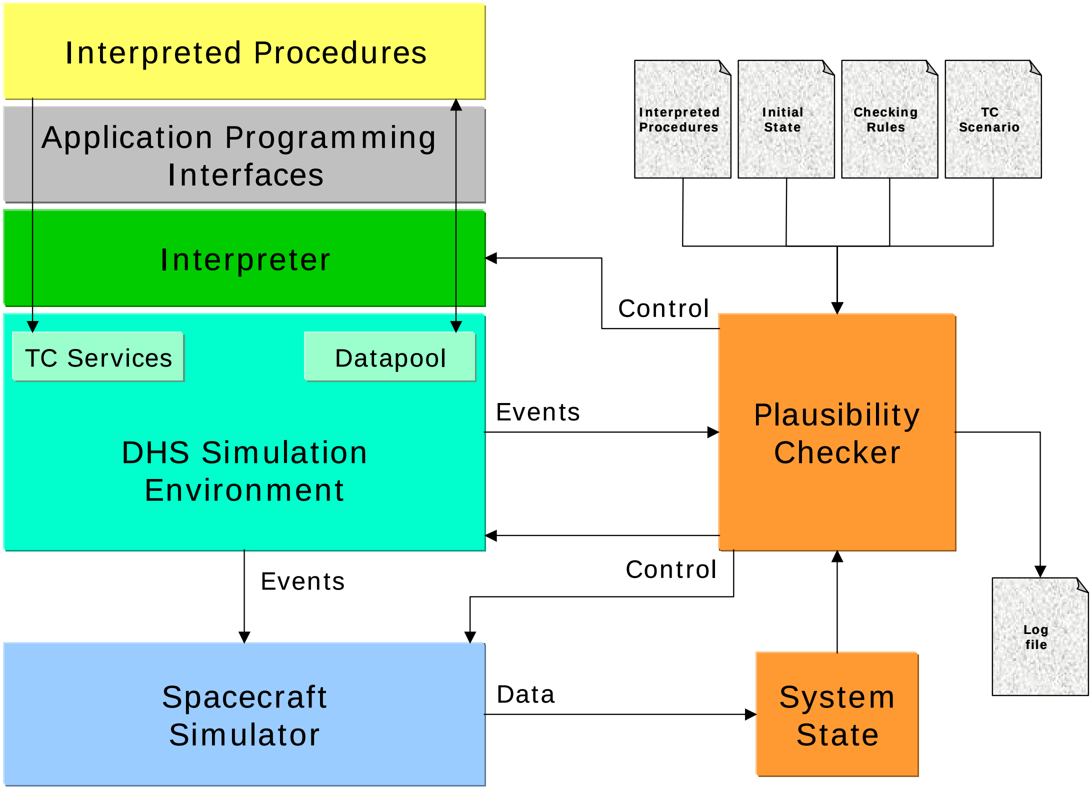
The survey of dependability and safety software issues for autonomy in space systems especially highlights:

- The importance of verification activities, which must be supported by various approaches and tools to widen the coverage for systems with such large state, input and behaviour spaces.
- Despite intensive verification and validation activities, there may remain design faults, as well as contexts and events leading to insufficiently specified and possibly inappropriate behaviours; consequently, it is necessary that mechanisms be provided to monitor possible anomalous situations and inappropriate behaviours when they occur, with the capability of maintaining as much as possible the desired properties, especially safety properties.

This leads to the definition of two kinds of software components for dependability and safety:

summary previous next

- A ground-based "plausibility checker" (Figure 2) to support and complement the ground validation of autonomy software, and especially the on-board control procedures before upload and actual execution.

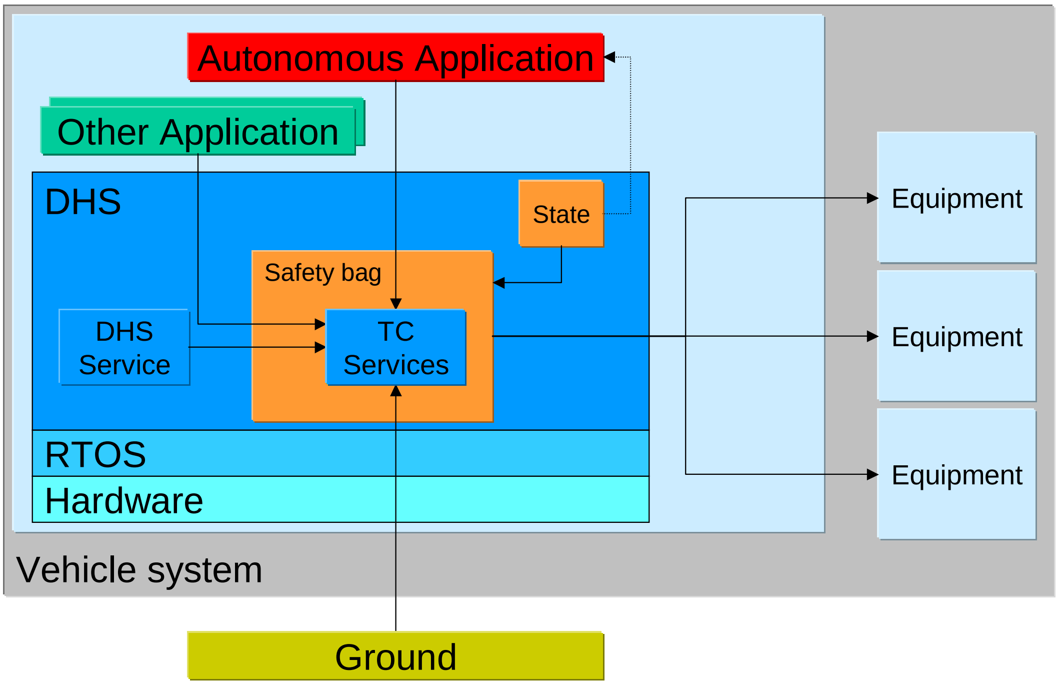


(DHS: Data Handling System; TC: Telecommand[1].)

Figure 2: Plausibility checker architecture and situation

- An on-board "safety bag" (Figure 3) 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.

[1] "Telecommand" is used in this paper as a generic term to designate the various commands sent to the equipment items on the platform or the payload, irrespective of their origin (ground or generated by an on-board application).



(DHS: Data Handling System; RTOS: Real-Time Operating System; TC: Telecommand[2])

Figure 3: Safety bag architecture and situation

The SPAAS project included the design and experimentation of the safety bag and plausibility checker software components, considered as generic components that can be instantiated and used in various real space projects with as few adaptations as possible, so as to support their dependability and safety.

6 Experimentation and Assessment

The safety bag and the plausibility checker were developed as generic components and their experimentation has been performed through a three-month pilot application on hardware, software and safety properties from real space projects. In this section, emphasis is placed mainly on the safety bag component.

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 validation 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

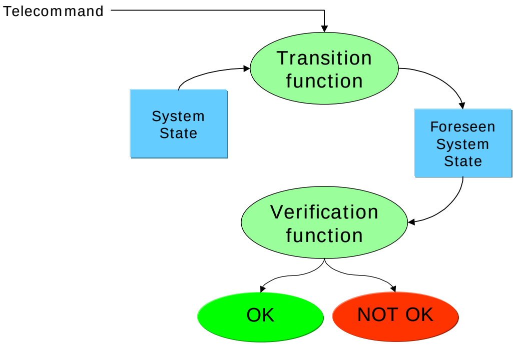
[2] As mentioned in note 1, “telecommand” designates any kind of command to an on-board equipment item, generated by the ground or by an on-board application. If all commands can be managed by the safety bag and potentially monitored according to a selected configuration, it is worth mentioning that the aim is mainly to monitor complex on-board software applications rather than transferring the ultimate responsibility from ground to board.

summary previous next

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 safety bag, on which the focus is put in this paper, 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:

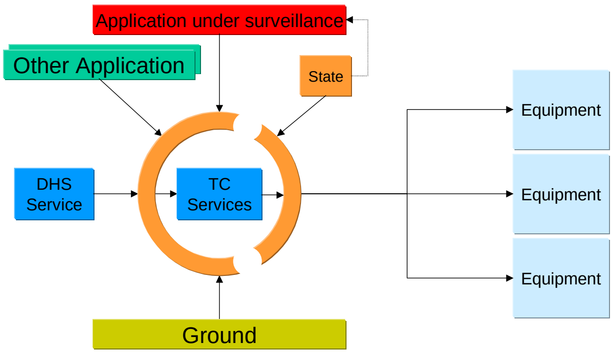
- The evaluation and assessment of performances (real-time performance and safety-related performance: coverage, latency, false alarm rate);
- The investigation of potential improvements or alternative solutions, particularly for the integration of the safety bag within the on-board platform architecture;
- 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 quite simple. 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.

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.



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.

summary previous next

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.

7 Conclusion

The study reported in this paper 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 insertion of the Safety Bag in an existing system is very simple. The major part of the work 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.

summary previous next

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 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.

8 References

[1] SPAAS project (Software Product Assurance for Autonomy on-board Spacecraft). Contract ESTEC 14898/01/NL/JA. SPAAS technical notes available at: ftp://ftp.estec.esa.nl/pub/tos-qq/qqs/SPAAS/StudyOutputs

[2] J. Fox and S. Das, “Safe and Sound - Artificial Intelligence in Hazardous Applications”, AIAA Press / The MIT Press, 2000.

[3] RIACS Workshop on the Verification and Validation of Autonomous and Adaptive Systems, 5-7 Dec. 2000, Asilomar Conference Center, Pacific Grove, CA: http://ase.arc.nasa.gov/vv2000/

[4] European Cooperation for Space Standardization (ECSS). Space Engineering — Software, ECSS-E-40B (draft 1), 29-5-2002, Space Product Assurance — Software Product Assurance, ECSS-Q-80B, 10-10-2003.

[5] F. Ingrand and F. Py, “An Execution Control System for Autonomous Robots”, in *IEEE International Conference on Robotics and Automation*, Washington, D.C., USA, 2002,

[6] D. Powell & P. Thévenod-Fosse, “Dependability Issues in AI-Based Autonomous Systems for Space Applications”, 2nd IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, October 7-8 2002, Toulouse, France, pp.163-177.

[7] P. Klein, “The Safety Bag Expert System in the Electronic Railway Interlocking System ELEKTRA”, *Expert Systems with Applications*, 3 (4), pp.499-560, 1991.