A Survey on Model-Based Mission Planning and Execution for Autonomous Spacecraft

Massimo Tipaldi and Luigi Glielmo, *Senior Member, IEEE*

***Abstract*—Different drivers are nowadays leading spacecraft toward an increased level of on-board autonomy. In this paper, we survey model-based techniques as a vehicle to implement highly autonomous on-board capabilities for spacecraft mission planning and execution. In this respect, spacecraft reconfiguration approaches based on Markovian Decision Process are explored, and then compared with other model-based alternatives. The integration of planning systems and dynamic reprogramming capabilities into the on-board software is presented. Finally, operational concepts for mission planning and execution in recent European space projects as well as the implementation of in-flight adaptive mission operations via on-board control procedures are also analyzed.**

***Index Terms*—Markov Decision Process (MDP), mission planning and execution, model-based spacecraft autonomy, on-board control procedures (OBCP), spacecraft operations, spacecraft reconfiguration.**

I. Introduction

IN CURRENT and future space applications, the need of designing spacecraft with a high level of on-board autonomy is emerging [1]. Its traditional notion as predefined and conservative sequences of on-board executed commands [2] can fulfill the needs for satellites operating in a predictable environment. However, this approach is not adequate for spacecraft operating in unpredictable contexts, which characterize deep-space exploration systems or critical operational phases, such as automated maneuvers for space rendezvous. Further drivers for increased on-board autonomy are the overall improvement of spacecraft availability and reliability [3], the need to overcome long communication delays and outages [4], and the reduction of costs in ground segment operations [5], which can address long-term planning instead of day-to-day procedures. This way, important requirements can be fulfilled, such as continuous mission product generation on board, real-time spacecraft control outside ground contact, maximization of mission objectives in relation to the available on-board resources, and robust operations in presence of on-board failures and context uncertainty.

Both the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA) regard on-board autonomy as a necessary technology for providing future space and interplanetary missions with cutting-edge science opportunities (see [1], [6], and [7]). Over the past decades, a certain degree of autonomy has been successfully integrated and tested within NASA space/interplanetary missions. We mention the following examples: NASA Deep Space One's (DS1's) Remote Agent (RA) [8], the first demonstrator of on-board autonomous planning/execution and fault management; the Continuous Activity Scheduling Planning Execution and Replanning [9], an iterative repair system used to update the current working plan in light of changing operating context; the Mixed-Initiative Activity Plan Generator [10], an autonomous mission planner used by Mars Exploration Rover; and the Autonomous Sciencecraft Experiment [11], an autonomous mission planner and executor flying on Earth Observing-1 (EO-1).

As far as ESA initiatives are concerned, we mention the following examples: the Aurora programme [12], whose objective is the robotic and manned exploration of the Solar System; the SMART-1 satellite [13], a technological demonstrator to experiment electrical propulsion for interplanetary missions and some concepts of satellite autonomy and ground automation tools; and PROBA [14], a mission designed to demonstrate the benefits of on-board autonomy. In PROBA, the importance of the on-board software (OBSW) in implementing on-board autonomy was already understood. The use of a higher level of autonomy in Rosetta spacecraft mission would have helped in dealing with some challenges, such as the navigation and prediction of comet activity [15]. In any case, some initial scientific operations of its lander Philae were carried out automatically with limited ground control [16].

From an operational point of view, on-board autonomy can be regarded as migration of functionality from the ground segment to the flight segment [6]. Jonsson *et al.* [1] identify four broad application fields, which are as follows.

1) Intelligent sensing: the ability to infer system state from the environmental sensor data.
2) Mission planning and execution: the process of decomposing high-level goals into a sequence of activities that satisfy temporal and resource constraints. An execution system is then responsible for dispatching such sequence of activities, while monitoring and reacting to off-nominal situations. As we can see, planning and execution are two different functions.
3) Fault management: given streams of observations, detecting, diagnosing, and reacting to events, and anomalies occurring inside a system.

Manuscript received October 4, 2016; revised January 10, 2017, April 6, 2017, and June 20, 2017; accepted June 22, 2017. Date of publication July 12, 2017; date of current version November 22, 2018. *(Corresponding author: Massimo Tipaldi.)*

M. Tipaldi is with the Department of Engineering, University of Sannio, Benevento 82100 Italy, and also with the OHB System AG/OHB Italia SpA, Bremen 28359 Germany (e-mail: mtipaldi@unisannio.it).

L. Glielmo is with the Department of Engineering, University of Sannio, Benevento 82100, Italy (e-mail: glielmo@unisannio.it).

Digital Object Identifier 10.1109/JSYST.2017.2720682

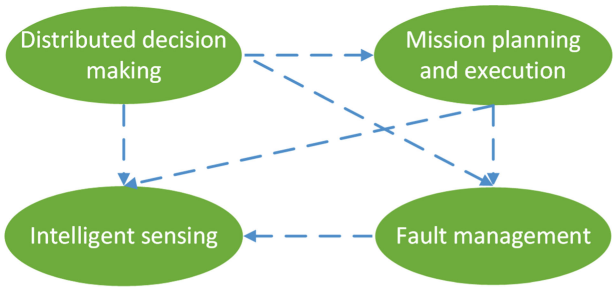

Fig. 1. Autonomy ecosystem for space missions.

4) Distributed decision making: effective cooperation among independent autonomous spacecraft in order to achieve common goals.

Fig. 1 gathers these application fields into a schematic view of the autonomy ecosystem for space missions. Dashed arrows between the application fields indicate an extension to functionality provided by the base application field. Intelligent sensing is a fundamental capability within the autonomy ecosystem. For instance, distributed decision-making applications can make use of the inferred state of satellite formations to achieve shared mission objectives [17]. As for mission planning and execution, future spacecraft are expected to be capable of receiving, processing, and achieving high-level objectives transmitted by the ground operators via ergonomic interfaces [18]. In this respect, we also consider updates to already uploaded plans. Ground high-level objectives can be further on-board detailed into a sequence of commands for the subsystems and can be autonomously adapted during their execution according to context changes, spacecraft health status, and altered on-board resource profile. On-board mission planning and execution driven by ground high-level objectives are relevant for remote planetary exploration missions, characterized by constrained communication links (frequency, transmission rates, and data volumes). In such missions, "Joysticking from Earth is no longer an option" [12]. Restrictions on uplink data rates and on the total number of commands to be sent to the spacecraft can be solved by sending single high-level commands, which can be disassembled on board to handle the spacecraft subsystems and units [19]. The addition of autonomous failure detection and recovery systems can reduce the number of mission outages, which occur in spacecraft characterized by low level of autonomous fault management. In such a case, the spacecraft enters a known safe configuration for each detected anomaly, while further investigations and recovery procedures can be carried out by the ground operators [20]. However, designing spacecraft with highly autonomous on-board capabilities should be balanced against other alternatives. In this respect, we can mention the Solar System Internetwork [21], a set of spatially distributed nodes, such as planetary stations and relay spacecraft. Such concept can address the limitations given by the simple point-to-point links between the spacecraft and Earth by providing robust and efficient end-to-end data services spanning terrestrial and space links on up to interplanetary distance scales.

In this survey, after outlining some aspects of interest in spacecraft autonomy (in particular, its state of art in recent European space projects), we focus on model-based approaches as a vehicle to implement highly autonomous on-board capabilities for mission planning and execution. Using a model-based approach means having on-board planners and controllers programmed with behavioral models of the spacecraft and its subsystems [2]. These models do not specify the sequence of actions required to fulfill specific high-level goals, but instead they define the expected effect each action or external event can have on the modeled system. Planners and controllers can use these models to synthesize sequences of actions directed toward the specified goals. As explained later in the paper, on-board reconfiguration is an important aspect for highly autonomous space systems and can be implemented via Markov Decision Process (MDP) based approaches. The latter can be used to provide spacecraft operating in unpredictable contexts with the capabilities of on-board execution for goal-oriented mission operations (as defined by the ground operators) and reactiveness to off-nominal conditions.

This survey has been organized as follows. This section and the next one provide an overview on spacecraft autonomy, its application fields, and relevant ESA/NASA projects with autonomy at their core. Section III describes the model-based software architecture of autonomous space systems. Section IV presents the operational approach for spacecraft mission planning and execution as well as the implementation of in-flight adaptive mission operations via on-board control procedures (OBCPs) in recent ESA projects. Autonomous spacecraft reconfiguration is addressed in Section V, where solutions based on MDP frameworks are explored. Some interesting links with robotics path planning are highlighted as well. Section VI analyzes other model-based approaches for spacecraft autonomy and compares them with MDP-based solutions. Finally, in Section VII, we draw conclusions.

II. Background on Spacecraft Autonomy

There is no consensus on the exact definition of autonomy and it is not simple to identify aspects of autonomous systems that can be invariant across different domains [22]. By analyzing the range of autonomous behavior of four case studies in space exploration missions, Jonsson *et al.* [1] highlight the following core needs for highly autonomous systems.

1) Autonomy describes a range of behaviors associated with agents, systems that can sense the world around them as well as their own state, can make decisions about what to do, and can carry out their decisions through their own action.
2) An autonomous system can be controlled by commanding it to achieve a set of goals; the system itself transforms the goals into sequences of actions that accomplish each goal.
3) An autonomous system flexibly responds to off-nominal situations by adjusting its activity sequence to attain the high-level goals and maintain system safety.

The ECSS-E-ST-70-11C standard [23] defines four levels of autonomy (see Table I, which also shows the sections of this

TABLE I
SPACECRAFT AUTONOMY LEVELS AS REPORTED IN [23]

Level	Description	Functions
E1	Mission execution under ground control; limited on-board capability for safety issues	Real-time control from ground for nominal operations; execution of time-tagged commands for safety issues (see Section IV)
E2	Execution of preplanned, ground-defined, mission operations on board	Capability to store time-based commands within an on-board scheduler (see Section IV)
E3	Execution of adaptive mission operations on board	Execution of on-board operations control procedures (see Section IV)
E4	Execution of goal-oriented mission operations on board	Goal-oriented mission replanning (see Sections III, V, and VI)

paper describing the solutions to implement such autonomy levels). Spacecraft can be endowed with a specific level of on-board autonomy, which determines the distribution of responsibilities between the ground segment and the spacecraft in accomplishing the space mission objectives [6]. In this respect, Esteve *et al.* [24] account for the following key-factors: mission type, mission objectives and priorities, type of spacecraft orbit, spacecraft ground visibility profile, operations concepts, and communication constraints. Three spacecraft categories can be identified: communication satellites, earth observation satellites, and spacecraft used in science and robotics exploration missions. Current European space missions typically use level E2, while some advanced earth observation and science missions implement level E3 [20]. Sequences of preplanned commands (level E2) are fairly simple in structure, and the execution by the OBSW is straightforward. The execution plan is a set of branching command sequences and subsequences, where each command is either executed at a specific time, or immediately following the completion of another command [25]. As a consequence of such "open-loop" scheme, dynamic outcomes and environmental uncertainties cannot be easily handled. Plans become inherently conservative [2]; for example, activities are assumed to take the longest they can possibly take and on-board failures can lead to abandoning the whole plan, whereas portions of the plan could still be safely continued. Level E3 adds more flexibility in the plan definition and execution (see Section IV). However, level E4 is the only one that meets the three above-mentioned needs and is the main topic of this paper.

Highly autonomous systems enable greater adaptability and lower operational cost for missions involving robotics, space systems with cooperative operations, and deep-space missions for planetary rendezvous, landing, sample mining and return. They become increasingly important as spacecraft are required to deal with phenomena that occur on time scales shorter than the communication latency between the spacecraft and Earth. However, adding such highly autonomous capabilities can be perceived as cost increase in space mission development and testing. Frost [3] highlights the relevant opportunities offered by on-board autonomy, and, as for cost mitigation, he recommends incrementing the autonomy capabilities on board the spacecraft gradually from one mission to the next one and leveraging on design and flight heritage.

Space agencies [1], including ESA [26], aim at increasing the autonomy level in future space missions. Achieving level E4 means designing a spacecraft able to make decisions based on a set of defined rules and goals, while autonomously replanning activities in case of off-nominal conditions. This presents much higher difficulties than levels E2 and E3, where the required decisions pertaining to a very limited number of conditions are defined and implemented *a priori*. Before implementing the full-blown level E4, Kucinskis *et al.* [26] propose to add a new intermediate level able to apply model-based reasoning for the prediction of potential on-board faults. The capability to take an autonomous action in case of prediction of potential faults can be still given to level E3. Such new autonomy level can pave the way for level E4, since it would give the opportunity to use and validate the model-based reasoning approach before dealing with replanning algorithms and their impact on the spacecraft operations.

For the sake of completeness and because of their direct interaction with the mission planning and execution (see Fig. 1), an overview of autonomy concepts applied to spacecraft fault management and distributed decision making in spacecraft constellations is provided in the remaining part of this section.

A. Autonomous Spacecraft Fault Management

Fault management, in general, but also particularly for spacecraft, is often referred to as FDIR, that is to say [27]: Fault Detection (the determination of the presence of faults in a system and their times of occurrence), Isolation (the determination of their location, type, and severity), and Recovery (the process of choosing the best action to recover from the anomaly condition). FDIR design is a very complex task and has to be regarded as a broad system-level activity [28]. It strongly depends on the spacecraft operational modes and mission phases. Basically, the FDIR system concept can be shaped between two extremes, i.e., having a straightforward FDIR (the spacecraft tends to enter a safe mode for each detected anomaly, the related recovery is completely performed by the ground, with the exception of vital functions whose anomalies are handled on board) or making use of a more sophisticated on-board FDIR concept (where faults are identified at the lowest level possible and solved autonomously by the spacecraft, thus keeping ground intervention to a minimum). In making this choice, there is a trade-off between mission outage versus costs in software FDIR capabilities and hardware redundancy.

Implementing autonomous spacecraft fault management means adopting a hierarchical FDIR architecture in order to solve faults at the lowest level and avoid their propagation. Faults are usually deployed along five hierarchical levels [18] (see Fig. 2) and are characterized by a specific severity, the function(s) involved in their detection, and the recovery sequence. The highest FDIR level is in charge of the correct execution of vital functions of the spacecraft, whereas lower level hierarchies operate at subsystem or unit level. A higher level is triggered by the adjacent lower level, only when the latter is not able to

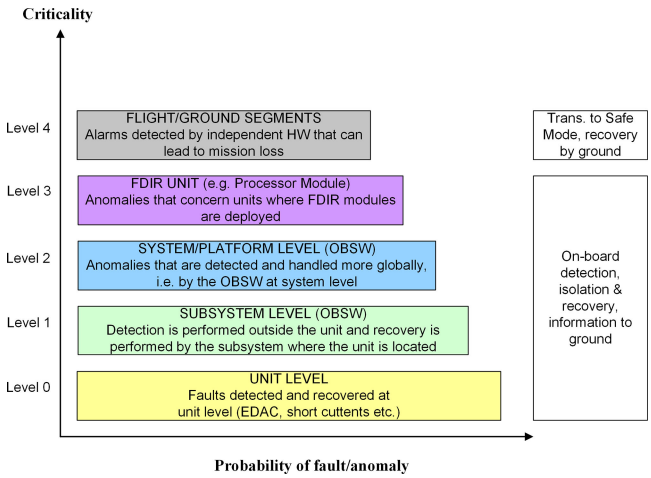

Fig. 2. Autonomous FDIR system hierarchical structure from [18].

solve/isolate a fault. In this case, FDIR functions allocated to the higher level can perform command and control functions of the next lower level in a certain sequence by using its housekeeping data and alarm information. The FDIR system hierarchical structure consists of the following levels.

1) Level 0: faults local to a unit that can be detected and recovered by local corrections, e.g., error detection and correction, short currents, overvoltage, and data bus failures. They have no impact on system performance.
2) Level 1: faults detected outside the unit and handled at subsystem level, e.g., anomalies related to unit communication interfaces. They can be recovered by resorting to the redundant path. As for the fault detection, the OBSW has to monitor unit data, e.g., against specific thresholds. Such faults can degrade subsystem performance.
3) Level 2: faults handled by the OBSW at system level, e.g., anomalies related to system-level functions caused by the propagation of undetected unit/subsystem faults. System-level data checks and functional monitoring characterize this level, whose anomalies can cause the loss of system-level functions and/or the degradation of their performance.
4) Level 3: anomalies related to the spacecraft on-board computer (OBC), which executes the central OBSW, thus level 1 and 2 FDIR functions. At this level, fault management consists of execution of OBC resets, reconfiguration procedures, and switches to safe mode as appropriate.
5) Level 4: anomalies not handled autonomously by the spacecraft. The latter executes specific reconfiguration procedures (according to patterns usually stored in a nonvolatile memory) and commands the OBSW to enter the safe mode for ground intervention. Anomalies at this level may lead to mission loss.

The higher the level, the more critical the faults but the lower their occurrence likelihood. Most of the detection and recovery actions are implemented in software, except for levels 0 and 4. The operational concept of a typical spacecraft includes one or more safe mode configurations. They represent the ultimate reaction to spacecraft severe anomalies and can be triggered either by on-board critical events (such as a severe spacecraft attitude excursion outside the operational range) or by ground. The spacecraft can remain in this mode without ground segment intervention for a specified period of time. In safe mode, the communication link to the ground segment, a specific power supply profile, and thermal survival functions for relevant equipment are maintained, whereas all nonessential on-board functions are powered off or disabled. The recovery of the spacecraft from safe mode to nominal mode needs to be commanded by ground.

Current FDIR approaches are actually based on quite stringent and rigid procedures [18]. Simple diagnostic routines usually process symptoms in isolation, which may result in incorrect diagnoses or contradictory deductions. As a consequence, they are not able to cope with the space environment, which is only partially observable by the FDIR monitoring function and is time variant. Research and development activities are nowadays focusing on model-based FDIR systems. They should provide the capabilities of processing anomalous observations in spite of uncertainties, system dynamic evolution, and partial observability in order to estimate the system health and determine the most appropriate corrective action. They can be combined with industrially consolidated FDIR approaches [18] with the aim of reducing the number of safe mode events, increasing the operational time of a spacecraft, and limiting the overall operational cost. For instance, model-based FDIR solutions can be applied to the level 2 of the FDIR hierarchical architecture (see Fig. 2) in order to improve the OBSW capabilities in fault identification at system level. Some surveys can provide a comprehensive overview on this topic. Marzat *et al.* [29] provide a classification of fault diagnosis quantitative and qualitative approaches (the former based on explicit mathematical models and control theories, the latter on artificial intelligence techniques) used in aerospace applications. Hwang *et al.* [30] also address various techniques of implementing reconfigurable control strategy in response to faults. They classify the reconfiguration techniques into the multiple-model approach, which uses a finite set of switching controllers, and the adaptive control approach, which changes the controller parameters in response to the detected faults.

B. Distributed Decision Making in Satellite Constellations and Federated Satellite Systems (FSSs)

Satellite clusters, that is to say autonomous and small satellites collaborating all together to achieve shared mission objectives, are becoming a new trend in spacecraft design thanks to the fact that they can offer higher performance, lower development costs, higher flexibility, better fault tolerance, and enhanced reconfigurability [31]. Multiagent systems could be designed, each agent with its own level of competence. Each satellite can integrate different types of agents according to its own role within the constellation. In this respect, Schetter *et al.* [17] present an example of agent-based software infrastructure and define the following functional agent categories: interaction agents, decision-making agents, organizational agents, representational agents, and operative agents. Each satellite can

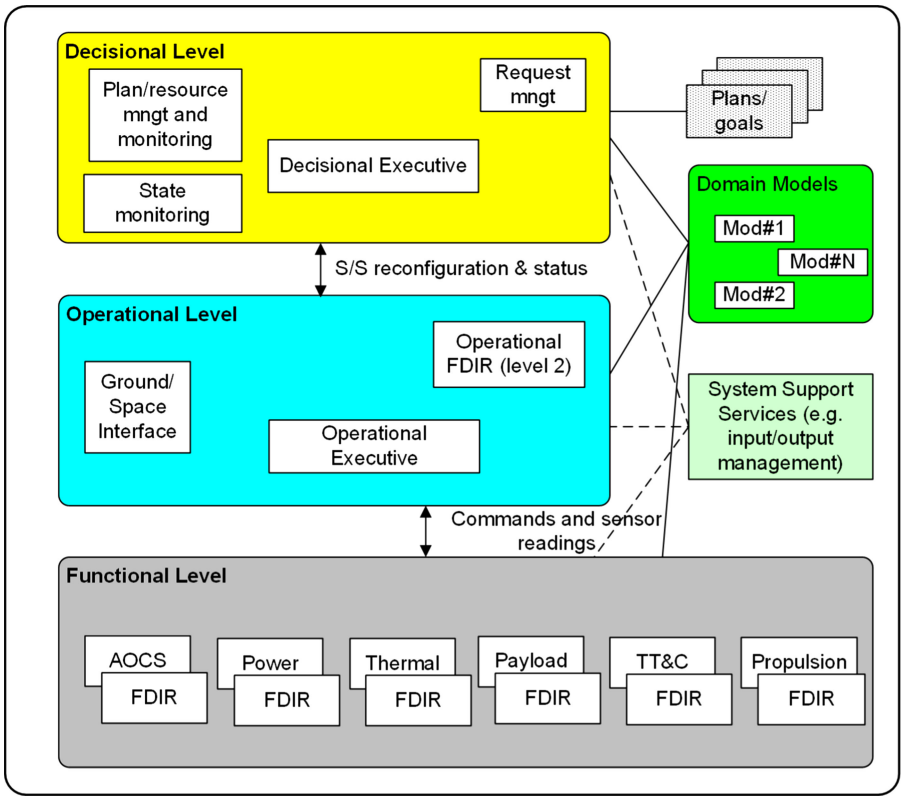

Fig. 3. Model-based OBSW architecture for autonomous space systems.

be classified according to the intelligence of its own allocated functional agents, while the overall constellation can be shaped from a rigid master-slave organization up to a fully distributed coordination architecture. Autonomous mission planner and activity scheduler for satellite clusters have to take into account both local and global objectives. Araguz *et al.* [31] propose an architecture with two levels of control: local and global. The purpose of the local level is to manage all the aspects that concern a single satellite, whereas the global level aims at handling the entire mission as a whole. Satellite formations can also offer additional challenges in the definition of the FDIR system architecture, since the formation has to be considered as an entity in itself. In other words, the FDIR hierarchical structure (see Fig. 2) is enriched by the global formation level, where FDIR requirements concerning functionality shared among different spacecraft are allocated [32]. When complex sequences of symptoms take place, the diagnosis becomes more difficult and diagnostics capabilities at formation level are needed in order to collect multiple anomaly reports, correlate them, and generate hypotheses explaining the observed anomalies [33].

In the literature, we can find other significant papers, for example the ones concerning FSSs, that is to say, networks of heterogeneous spacecraft with different goals and capabilities trading otherwise inefficiently allocated and unused resources, such as down-link bandwidth, storage capacity, processing power, and instrument time [34]. An FSS contrasts with most existing space-based systems, which are either monolithic (single spacecraft) or distributed but centrally managed (constellations). An FSS can be viewed as a System-of-Systems (SoS) and is characterized by a high degree of distributed authority among its components [35]. Each member of the satellite federation is called federate and is controlled by an independent entity, which decides its involvement based on localized value judgments. Architectural complexity is dominant in an FSS and the fact that each federate can actually choose between independent (noncooperative) and federated (cooperative) strategies poses additional challenges in the decision-making process [36].

III. Model-Based Software Architecture of Autonomous Space Systems

The OBSW plays a relevant role in implementing on-board mission planning and execution as well as fault management capabilities in autonomous space systems [18]. Accommodating level E4 autonomous capabilities within the OBSW is in line with the general trend of increasing OBSW complexity, supported by the growth in computer hardware performance [37]. The integration of such capabilities into the flight software is described in [38], where the OBSW architecture has been organized along three hierarchical levels (see Fig. 3).

1) The decisional level: it is in charge of programming spacecraft activity plans and monitoring their execution. They can be elaborated on board based on the objectives sent by ground and on the current state of the system (e.g., resource availability). An activity plan can be a controlled sequence of spacecraft/subsystems reconfigurations.

2) The operational level: it is in charge of the execution of the activity plans leading, e.g., to the subsystem reconfigurations as defined by the upper layer (decomposition and routing of commands). Ground can intervene at this level by sending direct telecommands (TCs) via a dedicated ground/space interface.
3) The functional level: it controls and supervises the various spacecraft subsystems (e.g., attitude and orbit control subsystem, power electrical subsystem, and thermal subsystem) by executing commands coming from the operational level and performing subsystem level monitoring.

Each layer performs a variation of a sense-think-act cycle or feedback control loop [1]. *Sensing* involves getting data from lower layers and mapping such information into a representation usable by the OBSW. *Thinking* involves considering the sensory data and information about the spacecraft, then arriving at what should be done to fulfill the desired objectives by using domain models. Finally, *acting* involves carrying out the decisions reached in the think cycle. These three layers can be distinguished in terms of information abstraction level, response-time requirement, and planning time resolution. The decisional layer provides the operational layer with abstract decisions and plans. The operational layer undertakes the relevant tasks of filling in such plans with more detailed sequences of commands and reacting when they do not lead to the expected results. In the functional layer, these commands are actually executed in order to bring the spacecraft into the desired state.

This approach implements the so-called model-based autonomy. On-board logic is programmed with spacecraft abstract models and on-board activity execution is dictated by high-level objectives and high sensitivity to the environment. A model-based approach basically implies the generation of sequences of reconfigurations and related control actions that move the physical plant along a desired evolution. Models of the plant are necessary to capture its nominal behavior and common failure modes [39]. In the case of autonomous space systems, given the ground high-level objectives and the spacecraft models, the decisional level defines the spacecraft operational mode and the reconfiguration sequence of its subsystems necessary to fulfill them. Thus, they are processed by the operational level, which generates a sequence of commands or control actions that can move the spacecraft subsystems into the desired state. This is again achieved by using the knowledge of the spacecraft/subsystem current states, their models, and the targeted states. At the core of both decisional and operational levels, the sequences of respectively reconfigurations and commands are generated by reasoning engines or state machine interpreters. These two levels are actually made up of two parts: the domain models and the engine (also called planner). The latter is a domain-independent item and performs the reasoning, that is to say a systematic exploration of the state space induced by the planning goals and the domain models. The sequence of onboard generated commands is executed by the functional layer.

Regardless of the layer, the sequence of commands or reconfigurations can be called policy. As a rule, information from the levels below is continually processed in order to confirm the successful transitions into the desired states. All three layers can be reactive, meaning that they respond immediately to changes in goals and to failures from their own perspective. Thanks to the model-based approach, goal-driven commanding is intertwined with fault detection, diagnosis, and recovery capabilities [40]. For instance, when a spacecraft subsystem strays from the specified state due to failures, the operational layer can analyze sensor data to identify the current status of the subsystems, and then move into a redundant reconfiguration (if any) in line with the desired goals.

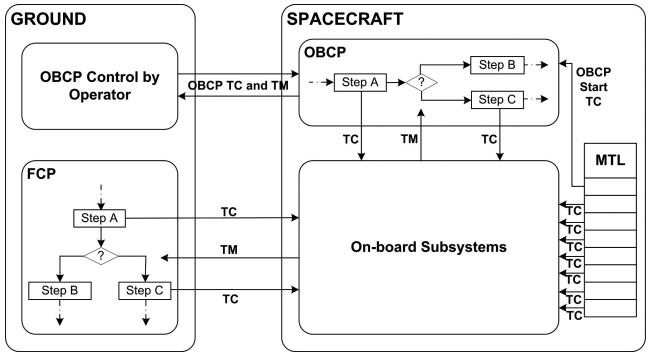

Fig. 4. Spacecraft operations: OBCP, FCP, and MTL as reported in [18].

To summarize, this section has shown how the model-based OBSW architecture of Fig. 3 can implement level E4 autonomy capabilities and how high-level objectives can be converted into commands, which can be actually executed by the spacecraft. We have also given emphasis on the controlled sequence of spacecraft/subsystems reconfiguration actions generated by the decisional layer. Finally, it is also worthwhile mentioning some operational aspects of model-based spacecraft autonomy. Policies should be designed during the spacecraft manufacturing phase, and then refined during the testing phase by means of ground segment software applications. Once calibrated, such policies should be somehow embedded into the OBSW and maintained during the spacecraft operational phase. This concept recalls the uploadable executable specification techniques as described in [41].

IV. Spacecraft Operations and OBCP in ESA Projects

In this section, we analyze the operational concepts for mission planning and execution in recent European space projects as well as the implementation of in-flight adaptive mission operations via OBCPs.

The ECSS-E-70-41A standard [25] defines the operational model of a spacecraft, which consists of a set of extensible services that can be invoked through a service request (telecommand (TC) source packet), such as invocation resulting in the generation of zero or more service reports (telemetry (TM) source packets). Spacecraft mission planning and execution can be basically performed by means of flight control procedures (FCPs), mission time line (MTL), and OBCPs [18] (see Fig. 4). FCPs are executed step-by-step by a ground operator, sending TC to the spacecraft and checking TM packets downloaded to ground. Missions with limited ground station coverage can also use the MTL

(autonomy level E2), which is a sequence of time-tagged TCs loaded from ground and executed by the OBSW when their time tag expires. OBCPs [42] (autonomy level E3) are self-standing procedures written as scriptlike files in a high-level programming language and compiled on ground into some efficient representation. After being uploaded, they are processed and executed by an on-board virtual machine (called OBCP interpreter), which is installed into the OBSW. OBCPs can be started via TCs or triggered by specific on-board events.

Steiger *et al.* [43] show the limitations of the MTL concept, since it is based on the assumption of successful commands (open-loop approach). On the contrary, OBCPs act as adaptive procedures (primitive closed-loop approach), therefore they increase spacecraft autonomy, reliability, and operability. Moreover, they can implement complex functional sequences (which are known in detail only at a later stage of the project) with the possibility of modifying them even during the mission. OBCPs can be updated and uploaded without requiring the uplink and the validation of the entire OBSW, because they are executed in a separate safe virtual machine. The OBCP development is completely independent from the OBSW apart from the fact that the services to load and execute OBCPs are needed. OBCPs are not only appropriate for nominal operations handling, but also suitable for more powerful on-board functionality developed during the extended mission phases. The following paragraph outlines the OBCP implementation and operational aspects as reported in different ESA missions, such as Rosetta [44], Venus Express [44], Herschel/Planck [45], Mars Express [46], BepiColombo [19], and Meteosat Third Generation Satellite [47].

A. OBCP Implementation and Operational Aspects in ESA Missions

OBCPs provide a useful mean to implement functional requirements as well as operational flight procedures, which can be modified, if necessary, even during the mission itself. They add flexibility during the later stages of the spacecraft testing activities, where it is usually necessary to tune some requirement implementation with limited nonregression tests due to project schedule pressure. OBCPs are flight control procedures that can be resident on board or can be uploaded to the spacecraft as required by ground. They serve for controlling spacecraft units and can be active for an extended period of time. The main use cases for OBCPs are as follows.

1) Spacecraft operations: OBCPs allow operations procedures to be executed on board as an alternative to execution from ground via FCPs. For instance, in Herschel/Planck operational procedures via OBCPs have been necessary due to the limited time of the spacecraft visibility from the ground segment [45].
2) Spacecraft availability and reliability: OBCPs can be used to implement autonomous fault management, for instance to rapidly react to on-board anomalies or events in case of deep-space missions [19].
3) Implementation of complex functional requirements: OBSW maintenance [43], spacecraft subsystem reconfiguration [44], mission-specific functions [47], and functionality seldom or even only once used, such as the spacecraft separation sequence in BepiColombo [19] or the entry and exit from the deep-space hibernation modes in Rosetta [44]. During the mission itself, additional OBCPs can be implemented, as happened in Mars Express, where they have secured science operations continuity after some anomalies in the solid state mass memory [46].
4) Testing automation of the OBSW up to the overall spacecraft: test configuration sequences, fault injection, and shortening of test execution time [47].

Thanks to their uses cases and design concept, OBCPs can be regarded as a way of implementing uploadable executable specification techniques [47]. Therefore, connections with some of the principles of model-based autonomy can be identified [41]. For instance, OBCP design concept is based on both domain-specific (i.e., the procedures) and domain-independent (i.e., the OBCP interpreter) components.

OBCP scripts are written in a high-level programming language (for instance derived from C [47] or ADA [44] languages). The OBCP language syntax and semantics are the result of a tailoring process of the OBCP language standard [42]. Compared to the MTL, important programming features are provided, such as the definition of user defined data types, the usage of control statements, and the declaration of functions/procedures (intermediate execution of special OBCPs with or without return value). Further features can be included to satisfy typical needs of space real-time applications, such as the capability of accessing the platform and payload TM data stored in the OBSW data pool and an in-built TM/TC interface to import data structures from the satellite reference database (SRDB). Fig. 5 shows an excerpt of OBCP as reported in [47].

After being designed, OBCPs are edited and compiled on ground to a compact binary token code. Before uploading and executing on board the spacecraft, they have to be verified and validated on ground [19]. For this purpose, the OBCP development environment (ODE) is used. Its main goal is to provide an environment to create and test OBCPs in a manner as consistent as possible with the procedure development interface more familiar to ground controllers and requiring only basic programming skills [46]. The ODE is normally composed of the following software components.

1) OBCP text editor, to write OBCP source code with shortcuts for programming language constructs and user-friendly retrieval of SRDB data structure definitions.
2) OBCP compiler, which checks the adherence of the OBCP script to the OBCP language syntax and to the SRDB definitions. The OBCP script is translated into the corresponding token code, which can be executed by the OBCP interpreter.
3) OBCP testing tool, which executes the stand-alone OBCP token code and offers OBCP debugging functionality.
4) Additional tools, such as a tool to compute bounds on the OBCP worst case execution time (WCET) and stack usage [47].

OBCP creation or modification require only the testing and up-link of the OBCP itself (and not the entire OBSW). The

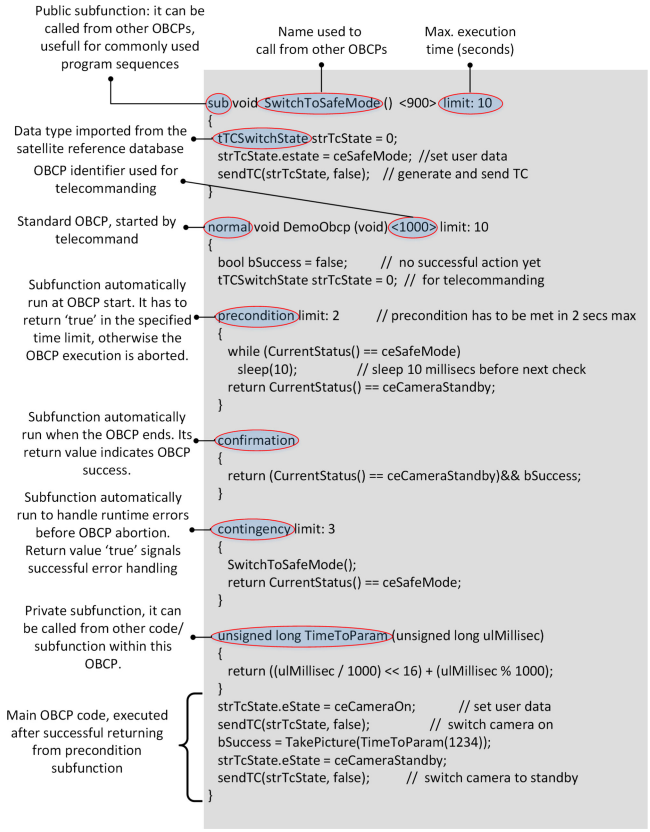

Fig. 5. OBCP procedure example as reported in [47].

OBCP life cycle is a tailored software life cycle, optimized to allow a straightforward development and test, adequate to the reduced complexity of the OBCPs, while keeping the strict SW version control as required by the OBSW. The OBCP verification phase can be seen as a module level test and aims at achieving its full branch coverage. The verification includes the analysis of static properties of the OBCP, such as its size, the expected stack occupation, and its WCET. Although each OBCP is executed on board into a safe and self-contained virtual machine, its execution must be preassessed on ground, including its compatibility with the available on-board resources (memory, CPU). To verify the dynamic behavior of the OBCP, a dedicated test environment replicating the on-board OBCP interpreter is employed: the OBCP testing tool. It consists of a stand-alone software program, which is used to execute OBCP token code in a simulated environment on ground, allowing certain visibility to the execution process and to the data maintained in the OBCP execution environment at run-time. Breakpoints can be set and cleared, memory contents checked and altered by the test conductor, and the OBCP execution can be stopped, resumed, or executed one token at a time. Moreover, the execution of each token is traced, so that the overall OBCP test coverage can be assessed once the test has run to completion. Finally, the OBCP can be validated in a more appropriate simulation environment, such as the software verification facility or H/W-in-the-loop test benches [48]. At this stage, if all the verification and validation criteria are passed, the OBCP is ready for its upload to the spacecraft.

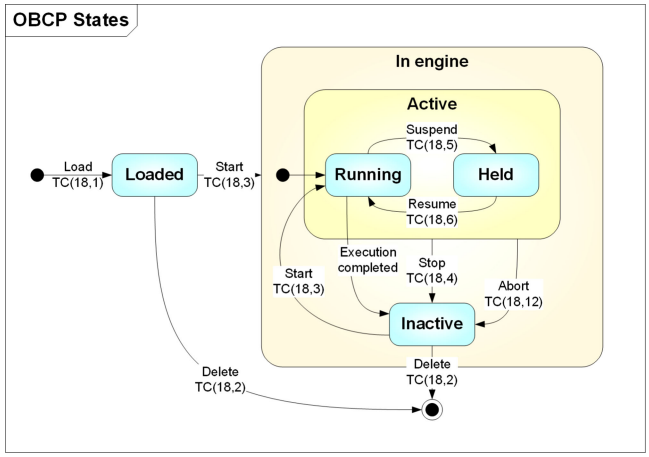

Fig. 6. OBCP state machine.

Once uploaded on board, OBCPs can evolve through different states (see Fig. 6). State transitions are performed by means of dedicated TCs, defined by tailoring the packet utilization standard services [25].

1) Before being executed, an OBCP needs to be loaded (loaded state) into a dedicated part of the OBC memory. Then, its execution can be started by a dedicated TC (running state).
2) As soon as its execution is completed, the OBCP enters the inactive state from which it can be either restarted or deleted.
3) During its execution, a TC can be sent to suspend the OBCP (held state). The OBCP execution can be resumed upon an user request.
4) Finally, the OBCP execution can be aborted (inactive state).

Some ESA missions (see [45] and [47]) provide the possibility of running more than one OBCP at the same time, the maximum number being defined at OBSW compile time. Each OBCP runs within a distinct OBCP interpreter instance and its execution is observable via spacecraft housekeeping TM.

V. Autonomous Spacecraft Reconfiguration

This section explores some approaches of the on-going debate on how to model spacecraft autonomy at level E4, which is quite an unexplored topic at both research and industrial level. In particular, we examine autonomous spacecraft reconfiguration, since it can be regarded as the cornerstone in mission planning and execution for autonomous spacecraft. As explained in Section III, given the ground high-level objectives, the whole spacecraft along with its subsystems automatically configures itself to fulfill them.

More specifically, we analyze MDP-based approaches as a vehicle to implement highly autonomous reconfiguration capabilities for spacecraft mission planning and execution. We also highlight some connections between spacecraft autonomy and robotics. Alternatives to MDP-based solutions for level E4 autonomous space systems are analyzed in Section VI.

A. Autonomous Spacecraft Reconfiguration via MDP

The main reason to formulate autonomous spacecraft reconfiguration as an MDP process can be found in its foundations [49]: spacecraft configurations as MDP states, stochastic transitions over the whole state space, and uncertainty in the outcome of actions. Policies can be calculated over the whole state space and well-proven solutions on how to solve MDP problems can be found in various spheres of application [50].

Realistic stochastic control problems are characterized by a large state space. Spacecraft are also affected by the so-called curse of dimensionality [51]. This issue calls for approximations to be performed on large-scale models. In this regard, approximate dynamic programming (ADP) techniques can be used to determine the suboptimal policies [52] over an MDP large-scale state space. Being applied to a wide range of contexts (spanning transportation, storage, financial, energy, military, medical, and manufacturing applications [53]), ADP techniques prove to be an emerging and powerful tool in tackling the curses of dimensionality and modeling for certain classes of multistage stochastic and dynamic problems [52]. Some examples of MDP approaches applied to spacecraft autonomous mission planning and execution are presented in the literature and discussed in this paragraph.

By providing some evidence on the improved performance for simulated planetary surface vehicles, Siddiqi *et al.* [49] highlight the importance of autonomous reconfiguration capabilities implemented via MDP-based approaches (see also [54], where the benefits of reconfigurability are measured in terms of functional availability via Markov-based reliability analysis). In particular, they propose a Nonhomogeneous Markov Chain [55] framework to incorporate context changes. State transition probabilities depend on some performance indices as well as on reconfiguration costs. Performance indices can be linked to external time-varying processes, such as the variation of soil properties and type. As the soil condition changes, the state probabilities converge after a few time steps toward new values, and the reconfiguration system can react to such context change by trying to enter the state where the probability is maximized.

Nasir *et al.* [56] use the discrete stochastic dynamic programming [50] to calculate the optimal sequences of attitude maneuvers for a spacecraft orbiting a planet and collecting data from a set of targets. The generated policy is optimal, meaning that the expected total reward of science data collected in the presence of the possible failures is maximized. Costs in performing actions are incorporated in the MDP transition probability matrix. The authors demonstrate how such costs and the discount factor of the formulated MDP framework can determine the shape of the optimal policy.

In a more recent paper, Nasir *et al.* [51] present an MDP framework to compute postfault optimal reconfiguration policies based on the remaining mission objectives to be achieved, the actions currently performed by the spacecraft, and the fault flag vectors generated by the on-board fault detection mechanism, including their probabilities of correctness. By considering no uncertainty in state transitions, the MDP problem can be solved by maximizing an objective function, which incorporates the above-mentioned factors. This approach is applied to a case study inspired by a real space mission, where the proposed fault reconfiguration framework would have avoided the loss of substantial mission time with the spacecraft in safe mode while awaiting ground reconfiguration instructions. The authors face the computational complexity associated with the large state-space by splitting the original problem into smaller tractable ones. In particular, they exploit simple structural properties of the system, that is to say the reconfiguration of the gyroscopes can be separated from the ones of the control laws. It is also highlighted how the framework can select dangerous options if the parameters in the objective function are not tuned carefully.

The authors of this survey present an MDP-based framework as a way of modeling on-board autonomy mechanisms for mission planning and execution [57]. It is applied to the decisional layer of the autonomous space systems architecture and reuses some ideas from [51]. In order to solve the curse of dimensionality, it employs ADP feature-based state aggregation techniques [52] to determine the suboptimal policies over an MDP large-scale state space. The MDP state space is partitioned based on its reward function structure, and the optimal cost-to-go or value function is approximated by a constant over each partition or metastate. The justification of this choice is quite straightforward. Information coming from lower levels can be easily gathered into aggregate states within the decisional level, and this can lead to a reduction of the formulated MDP dimension. At decisional level, we are more interested in compound information and actions, rather than in specific aspects of the spacecraft and its subsystems. For instance, at this level, what is really important is not the peculiarity of a subsystem report, but its severity, which is actually the driver of the action to be chosen. As a result, we can define an MDP-based framework for constructing an autonomous spacecraft reconfiguration mechanism that is optimal with respect to maximizing the achievement of mission objectives in the various mission phases. It also minimizes the effects of failure and accounts for the possibility of unsuccessful reconfiguration action. The decision-making process basically involves choosing the best reconfiguration policy, which implies executing a set of tasks at the operational level.

Both [51] and [57] as well as [56] highlight the importance of defining the reward function structure and choosing the values of its parameters. This requires some theoretical background on MDP (and ADP) as well as some practical insight into the problem to be solved. They influence the shape of the calculated (sub)optimal policy, which should fulfill project requirements, for example risking the spacecraft safety for the sake of some really important mission objectives, such as collecting data from an event that occurs once in a very long time.

B. Robotics and Spacecraft Autonomy

It is interesting to acknowledge relevant similarities between spacecraft autonomy and robotics from a problem formulation and resolution perspective. In this paragraph, we highlight how robotics motion planning can be a relevant source of information and ideas in implementing spacecraft autonomous mission planning and execution. MDPs are widely used in robotics, and

issues such as the curses of modeling and dimensionality affect their application to practical cases as well. A few examples are reported along with their connection to the concepts related to spacecraft autonomy.

Cheng *et al.* [58] propose a machine-learning framework for e-pet, an animal-type robot companion. A three layered hierarchical architecture is adopted and includes the instinct level, the perception level, and the planning level. They are very similar to the ones described in Section III. Based on some domain knowledge, the planning level disassembles high-level instructions into a sequence of reconfigurations for the perception level, where an MDP is adopted. Afterward, a sequence of more detailed actions is created and forwarded to the instinct level, which makes sure that the agent does not come into risky contact with the environment. The e-pet learns by interacting with humans using reinforcement leaning techniques in order to update the MDP model and the domain knowledge.

As for the curse of dimensionality, Corona-Xelhuantzi *et al.* [59] present an approach for partitioning complex MDPs into simpler ones via functional decomposition. The optimal policy for each subMDP is calculated, and then combined in order to solve resource and behavioral conflicts. Local policies are run on-line by applying the set of restrictions derived from the conflict analysis. The functional decomposition is natural in robotics. For example, we can identify navigation, vision, interaction, and manipulation functions, each contributing to common objectives. The same can be said for spacecraft. The corresponding local policies have to be run in parallel leading to resource and behavioral conflicts, for instance spacecraft have to turn their solar panels to face the sun and charge their batteries to a certain level before capturing some images from a camera. Bakker *et al.* [60] use abstraction to create hierarchical MDP models, where states with similar features are grouped together for robots path planning. The calculation of (sub)optimal policies for each level of abstraction is more efficient, even though this means having some extra memory and overhead to represent and handle the hierarchical system. Feature-based abstraction can be a way of modeling autonomous space systems as shown in Section V. Moreover, problem domain knowledge and problem specific heuristics can alleviate the computational burden of the MDP solvers [61].

Q-learning [62] can be used to counteract the prohibitive tasks of modeling robots path planning via MDP. And since spacecraft are SoS [63], they are also affected by the so-called curse of modeling. We mention only two significant examples for mobile robot navigation problem. Jaradat *et al.* [64] manage to apply the Q-learning algorithm for a mobile robot path planning in a dynamic environment by limiting the number of states through aggregation of system states into safe states, nonsafe states, winning states and failure states. Zuo *et al.* [65] propose a hierarchical approach made up of two levels. In the first level, the A^* algorithm [66] is used to find a near optimal geometric path. A set of subgoals is given to the second level, where the least-square policy iteration (LSPI) [67] is applied to smooth and optimize the path. By combining different effective techniques, such as value function approximation and improvement of policies by exploiting samples collected off-line from the simulated or even the real process, the LSPI is a Q-learning-based approach and provides a very interesting methodology with better properties in convergence and stability than other reinforcement learning algorithms [68].

VI. Comparison With Other Model-Based Approaches

In this section, we present other model-based approaches for mission planning and execution of level E4 autonomous space systems, and then we compare them with MDP-based solutions.

The first significant example of spacecraft autonomy methodology dates back to the NASA DS1 technology validation mission, where the RA flight software was used to generate and execute a plan to accomplish the goals in a robust manner [8]. The RA architecture is similar to the one described in Section III. It is based on the Livingstone model-based diagnosis system [69] and uses a qualitative, discrete, propositional logic based, reusable inference engine, which accepts a model of the system, such as a spacecraft, and keeps track of the commands to and the observations from the system. By using them, it can monitor the system and diagnose its current state. Plan execution robustness has been added to the Livingstone framework by making use of the mode identification and reconfiguration (MIR) system. The MIR can be regarded as a discrete model-based controller with MI providing the sensing of component modes and MR playing the role of actuator. The MI part uses a conflict-direct best-first search to find the most likely combination of components modes in line with the observed low-level sensor data, whereas the MR part uses the same search to find the least-cost combination of reconfiguration actions that achieve the desired goals. Over the years, Livingstone has had some successful applications [70]. Williams *et al.* [39] present another framework for model-based programming of robotic space explorer: it is based on Titan, which is an evolution of the Livingstone engine.

Another example of model-based spacecraft autonomy can be found in [71], where the autonomous reasoning engine (ARE) is proposed. The ARE integrates important functionality (such as plan generation, plan execution and monitoring, FDIR, and run-time diagnosis) in a uniform framework. Such framework is used to synthesize a plan from the formal models of the spacecraft (and its environment) and the logical specification of the objectives to be achieved. It is based on the approach known as planning as symbolic model checking [72]. Formal models describe both the nominal and the degraded/faulty behavior of the spacecraft. The ARE implements the architecture described in Section III. Reachable states and contingency plans (the latter to incorporate the nondeterminism of the environment and the partial observability of the spacecraft) are generated. Based on the information coming from the functional layer, the operational layer can execute the proper branch within the contingency plan. The decisional layer also includes an FDIR block, which is activated in response to an anomaly detected by the operational layer. It decides whether to continue the execution of the remaining part of the plan, or to attempt a replanning, or to move into a safe mode.

Further approaches used to implement model-based spacecraft autonomy are reported hereafter.

1) Indra *et al.* [73] propose a hierarchical architecture along three different planning horizons, each characterized by an appropriate knowledge model, that is to say the goal-task structure model, the abstract spacecraft model, and the detailed spacecraft model. Time Petri nets are used in all the three layers.
2) Vassev *et al.* [74] present an approach for self-scheduling mechanism for NASA swarm-based exploration missions. It uses the autonomic system specification language, a multitier specification model to express and specify self-configuring, self-healing, and safety properties of NASA swarm-based missions.
3) Ruia *et al.* [75] propose a software architecture for deep-space explorer mission planning based on multiagent systems. All the planning agents communicate and cooperate with each other to produce partial plans that satisfy all the constraints of the system. The formalism used therein to represent the planning problem is interesting.

The real benefit of using MDP-based frameworks versus Livingstone model-based diagnosis systems is that the former allow most of the calculation off-line to determine the optimal policy, whereas the latter use on-line calculation by adopting conflict-directed best-first search [51]. MDP-based optimal policy generation can be performed off-line, with the output being a comprehensive policy, then executed on board of the spacecraft [56]. Other benefits come from the usage of MDP-based approaches in a great deal of applications with the possibility of reusing well-proven solutions to calculate the (sub)optimal policies. For example, simulation-based ADP approaches [52] could be suitable candidates, where the value function of a given policy or even the optimal value function can be approximately computed through simulated trajectories, which can be generated during the spacecraft test campaign. Key benefit of simulation is also the possibility to perform high-dimensional linear algebra calculations by using vectors in low-dimensional feature space [52]. Moreover, by applying Q-learning approaches, the knowledge of the spacecraft model is not necessary. Simulation-based ADP and Q-learning approaches are important to scale up MDP formulation to real applications [53].

MDP and model checking take different approaches to deal with uncertainties of the planning problem [72]. With MDPs, we have probabilities, a reward function, and an optimization problem. With "planning as model checking," we have nondeterministic state transition systems, a logical specification of the goal, and a satisfiability problem. In MDP-based solutions, planning is performed through the construction of optimal policies with respect to rewards and probability distributions, which are often difficult to define due to the lack of enough statistical data [50].

As for model-checking approaches, solutions are defined in terms of reachable states and contingency plans. Model checking also suffers from the state space explosion, which can be solved by expensive discrete abstractions [63]. The main advantage of model checking is the expressiveness of temporal logic formula in specifying mission objectives [72]. On the other hand, it is not straightforward to force an MDP-based planner to generate a solution that satisfies some logical requirements [72]. However, in very recent papers, we have found promising attempts in learning optimal policies or synthesizing controllers via ADP [76] or Q-learning [77], which satisfy temporal logic specifications. They can pave the way for interesting applications in the planning problem by exploiting the benefits and counterbalancing the flaws of these two methodologies.

VII. Conclusion

In this paper, after having described the solutions used in recent European space projects for implementing on-board autonomy, we have surveyed model-based and MDP approaches as a vehicle to implement highly autonomous on-board capabilities for space mission planning and execution.

In particular, we have addressed the highest level of on-board autonomy according to the European space engineering standards, the level E4, which means designing spacecraft able to make decisions based on a set of goals and to replan activities in case of off-nominal conditions. We have also explained how to accommodate level E4 autonomous capabilities within the three layered OBSW architecture. In this respect, we have demonstrated that spacecraft reconfiguration is the underlying principle of autonomous space systems, and that it can be implemented via model-based and MDP approaches.

In the final part of the paper, we have acknowledged interesting similarities between spacecraft autonomy and robotics. This can be very encouraging since robotics can be a relevant source of information and ideas in implementing spacecraft autonomous mission planning and execution. However, based on the analysis of the referenced literature and on our own experience, we venture the following remarks. Even though there is a tangible need for having highly autonomous spacecraft, there is still a lot to do at both concept and design level. For instance, it is really difficult to identify a well-established conceptual approach for spacecraft autonomy. This has been highlighted when we have compared MDP with other model-based solutions, such as planning as model checking. Both have advantages and disadvantaged, and there are some very recent attempts in the literature to combine them. As a result, the technological readiness level is still not adequate to design and implement highly autonomous capabilities in real space missions. The actual challenge is to define the approach for calculating policies at operational level (of the three layered OBSW) and their links with the policies defined at decisional level in real projects. Finally, prototypes of the decisional and operational layers (including ground segment software applications for policy calculation and upload) have to be implemented and the corresponding spacecraft operations concepts defined in more detail.

References

[1] A. Jonsson, R. Morris, and L. Pedersen, "Autonomy in space current capabilities and future challenges," *AI Mag.*, vol. 8, no. 4, pp. 27–42, 2007.
[2] V. Verma, A. Jonsson, R. Simmons, T. Estlin, and R. Levinson, "Survey of command execution systems for NASA spacecraft and robots," in *Proc. Int. Conf. Automat. Planning Scheduling*, 2005, pp. 92–99.
[3] C. R. Frost, "Challenges and opportunities for autonomous systems in space," *Nat. Acad. Eng. U.S. Frontiers Eng. Symp.*, pp. 1–17, 2010.

[4] R. Sterritt, C. A. Rouff, M. G. Hinchey, J. L. Rash, and W. Truszkowski, "Next generation system and software architectures challenges from future NASA exploration missions," *Sci. Comput. Program.*, vol. 61, pp. 48–57, 2006.

[5] J. van der Ha, "Trends in cost-effective mission operations," *Acta Astronaut.*, vol. 52, no. 2, pp. 337–342, 2003.

[6] T. Grant, A. O. Soler, A. Bos, U. Brauer, M. Neerinex, and M. Wolff, "Space autonomy as migration of functionality: The MARS case," *Proc. Space Mission Challenges Inf. Technol.*, pp. 201–208, 2006.

[7] J. A. Starek, B. Acikmese, I. A. D. Nesnas, and M. Pavone, "Spacecraft autonomy challenges for next generation space missions," *Advances in Control System Technology for Aerospace Applications* (Lecture Notes in Control and Information Sciences), vol. 460, E. Feron, Ed. Belin, Germany: Springer-Verlag, 2015, ch. 1, pp. 1–48.

[8] D. E. Bernard *et al.*, "Design of the remote agent experiment for spacecraft autonomy," in *Proc. IEEE Aerosp. Conf.*, 1998, pp. 259–281.

[9] S. Chien, A. Knight, R. Stechert, R. Sherwood, and G. Rabideau, "Using iterative repair to increase the responsiveness of planning and scheduling for autonomous spacecraft," in *Proc. 5th Int. Conf. Artificial Intell. Planning Syst. (AIPS'00)*, 1999, pp. 300–307.

[10] M. Ai-Chang *et al.*, "MAPGEN: Mixed initiative planning and scheduling for the Mars '03 MER Mission," in *Proc. Seventh Int. Symp. Artificial Intell., Robot. Autom. Space*, 2003.

[11] R. Sherwood *et al.*, "The EO-1 autonomous sciencecraft," in *Proc. Small Satell. Conf.*, 2007, pp. 1–9.

[12] M. J. Woods, R. S. Aylett, D. P. Long, M. Fox, and W. Roger, "Developing autonomous AI planning and scheduling technologies for remote planetary exploration," in *Proc. 7th ESA Workshop Adv. Space Technol. Robot. Autom.*, 2002, pp. 1–7.

[13] O. Caminoa *et al.*, "SMART-1 operations experience and lessons learnt," *Acta Astronaut.*, vol. 61, pp. 203–222, 2007.

[14] J. Bermyn, "PROBA—Project for on-board autonomy," *Air Space Eur.*, vol. 2, no. 1, pp. 70–76, 2000.

[15] M. Ashman *et al.*, "Rosetta science operations in support of the Philae mission," *Acta Astronaut.*, vol. 125, pp. 41–64, 2016.

[16] P. Di Lizia *et al.*, "Planning and implementation of the on-comet operations of the instrument SD2 on board the lander Philae of Rosetta mission," *Acta Astronaut.*, vol. 125, pp. 183–195, 2016.

[17] T. Schetter, M. Campbell, and D. Surka, "Multiple agent-based autonomy for satellite constellations," *Artif. Intell.*, vol. 145, no. 1/2, pp. 147–180, 2003.

[18] M. Tipaldi and B. Bruenjes, "Survey on fault detection, isolation, and recovery strategies in the space domain," *J. Aerosp. Inf. Syst.*, vol. 12, no. 2, pp. 235–256, 2015.

[19] A. Schwab, R. Eilenberger, and W. Zur Borg, "OBCPs—An integrated part of the BepiColombo autonomy and flexibility," in *Proc. 12th Int. Conf. Space Oper.: SpaceOps*, 2012.

[20] X. Olive, "FDI(R) for satellites: How to deal with high availability and robustness in the space domain?," *Int. J. Appl. Math. Comput. Sci.*, vol. 22, no. 1, pp. 99–107, 2012.

[21] C. D. Edwards, M. Denis, and L. Braatz, "Operations concept for a Solar System Internetwork," in *Proc. IEEE Aerosp. Conf.*, 2011, pp. 1–9

[22] D. P. Watson and D. H. Scheidt, "Autonomous systems," *Johns Hopkins APL Tech. Dig.*, vol. 26, no. 24, pp. 368–376, 2005.

[23] *Space Engineering—Space Segment Operability*, European Cooperation for Space Standardization Standard ECSS-E-ST-70-11C, 2008.

[24] M. Esteve, J. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein, "Formal correctness, safety, dependability, and performance analysis of a satellite," in *Proc. Int. Conf. Softw. Eng.*, 2012, pp. 1022–1031.

[25] *Ground Systems and Operations—Telemetry and Telecommand Packet Utilization*, Eur. Cooperation for Space Standardization ECSS-E-70-41A, 2003.

[26] F. Kucinskis and F. Ferreira, "Taking the ECSS autonomy concepts one step further," in *Proc. SpaceOps 2010 Conf.*, 2010, pp. 1–9.

[27] A. Wander and R. Forstner, "Innovative fault detection, isolation and recovery on-board spacecraft: Study and implementation using cognitive automation," in *Proc. Conf. Control Fault-Tolerant Syst.*, 2013, pp. 336–341.

[28] L. Fesq, "Current fault management trends in NASA's planetary spacecraft," in *Proc. IEEE Aerosp. Conf.*, 2009, pp. 1–9.

[29] J. Marzat, H. Piet-Lahanier, F. Damongeot, and E. Walter, "Model-based fault diagnosis for aerospace systems: A survey," *Proc. Inst. Mech. Eng., Part G, J. Aerosp. Eng.*, vol. 226, no. 10, pp. 1329–1360, 2012.

[30] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 3, pp. 636–653, May 2010.

[31] C. Araguz, A. Alvaro, I. del Portillo, K. Root, E. Alarcon, and E. Bou-Balust, "On autonomous software architectures for distributed spacecraft: A local-global policy," in *Proc. IEEE Aerosp. Conf.*, 2015, pp. 1–9.

[32] C. Castel, J. Gabard, and C. Tessier, "FDIR strategies for autonomous satellite formations—A preliminary report," in *Proc. Symp. Spacecr. Auton., Using AI Expand Human Space Explor.*, 2006, pp. 1–8.

[33] A. Dubey *et al.*, "A software platform for fractionated spacecraft," in *Proc. IEEE Aerosp. Conf.*, 2012, pp. 1–20.

[34] A. Golkar and I. Lluch, "The Federated satellite systems paradigm: Concept and business case evaluation," *Acta Astronaut.*, vol. 111, pp. 230–248, 2015.

[35] P. T. Grogan, A. Golkar, S. Shirasaka, and O. L. de Weck, "Multi-stakeholder interactive simulation for federated satellite systems," in *Proc. IEEE Aerosp. Conf.*, 2014, pp. 1–15.

[36] P. T. Grogan, K. Ho, A. Golkar, and O. L. de Weck, "Multi-actor value modeling for federated systems," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1193–1202, Jun. 2018.

[37] R. Butler and M. Pennotti, "The evolution of software and its impact on complex system design in robotic spacecraft embedded systems," *Conf. Syst. Eng. Res.*, vol. 16, pp. 747–756, 2013.

[38] S. Lemai, X. Olive, and M. Charmeau, "Decisional architecture for autonomous space system," in *Proc. ESA Workshop Adv. Space Technol. Robot. Autom.*, 2006, pp. 1–8.

[39] B. C. Williams, M. Ingham, S. Chung, and P. Elliott, "Model-based programming of intelligent embedded systems and robotic space explorers," *Proc. IEEE*, vol. 91, no. 1, pp. 212, 237, 2003.

[40] L. M. Fesq, M. D. Ingham, M. Pekala, J. van Eepoel, and B. C. Williams, "Model-based autonomy for the next generation of robotic spacecraft," in *Proc. Int. Astronaut. Congr., Int. Astronaut. Feder.*, 2002, pp. 1–10.

[41] G. Cancro, W. Innanen, R. Turner, C. Monaco, and M. Trela, "Uploadable executable specification concept for spacecraft autonomy systems," *Proc. IEEE Aerosp. Conf.*, 2007, pp. 1–12.

[42] *Space Engineering—Spacecraft On-Board Control Procedures*, Eur. Cooperation Space Standardization ECSS-E-ST-70-01C, 2010.

[43] C. Steiger, J. Morales, and R. Furnell, "OBSM operations automation through the use of on-board control procedures," in *Proc. 8th Int. Conf. Space Oper.*, 2004, pp. 1–10.

[44] C. Steiger, J. Morales, and R. Furnell, "On-board control procedures for ESA's deep space missions Rosetta and Venus express," in *Proc. DASIA Data Syst. Aerosp. Conf.*, 2005, pp. 1–12.

[45] M. Ferraguto, T. Wittrock, M. Barrenscheen, M. Paakko, and V. Sipinen, "The on-board control procedures subsystem for the Herschel and Planck satellites," in *Proc. 32nd Annu. IEEE Int. Comput. Softw. Appl.*, 2008, pp. 1366–1371.

[46] D. T. Lakey *et al.*, "Multi-mission end-to-end OBCP configuration control," in *Proc. 12th Int. Conf. Space Oper., SpaceOps*, 2012.

[47] M. Tipaldi *et al.*, "Spacecraft autonomy and reliability in MTG satellite via on-board control procedures," in *Proc. IEEE Int. Workshop Metrol. Aerosp.*, 2015, pp. 155–159.

[48] R. Hendricks and J. Eickhoff, "The significant role of simulation in satellite development and verification," *Aerosp. Sci. Technol.*, pp. 273–283, 2005.

[49] A. Siddiqi, O. L. De Weck, and K. Iagnemma, "Reconfigurabilty in planetary surface vehicles: modeling approaches and case study," *J. Brit. Interplanet. Soc.*, vol. 59, pp. 450–460, 2006.

[50] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: Wiley, 1994.

[51] A. Nasir, E. Atkins, and I. Kolmanovsky, "A mission based fault reconfiguration framework for spacecraft applications," in *Proc. Infotech@Aerosp. Conf.*, 2012, pp. 1–12.

[52] D. P. Bertsekas, "Approximate policy iteration: a survey and some new methods," *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310–335, 2011.

[53] W. B. Powell, "Approximate dynamic programming for high-dimensional problems," in *Proc. Winter Simul. Conf.*, 2007, pp. 1–11.

[54] A. Siddiqi and O. L. De Weck, "Reconfigurability in planetary surface vehicles," *Acta Astronaut.*, vol. 64, no. 5/6, pp. 589–601, 2009.

[55] W. Feller, "An introduction to probability theory and its applications volume I," 3rd ed. New York, NY, USA: Wiley, 1968.

[56] A. Nasir, E. Atkins, and I. Kolmanovsky, "Science optimal spacecraft attitude maneuvering while accounting for failure mode," in *Proc. 18th World Congr. Int. Fed. Automat. Control*, 2011, pp. 812–817.

[57] M. Tipaldi and L. Glielmo, "State aggregation approximate dynamic programming for model-based spacecraft autonomy," in *Proc. Eur. Control Conf.*, 2016, pp. 86–91.
[58] S. Cheng, T. Chang, and C. Hsu, "A framework of an agent planning with reinforcement learning for e-pet," in *Proc. Int. Conf. Orange Technol.*, 2013, pp. 310–313.
[59] E. Corona-Xelhuantzi, E. F. Morales, and E. Sucar, "Executing concurrent actions with multiple Markov decision processes," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn.*, 2009, pp. 82–89.
[60] B. Bakker, Z. Zivkovic, and B. Krose, "Hierarchical dynamic programming for robot path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 2756–2761.
[61] C. Meirina, Y. N. Levchuk, G. M. Levchuk, and K. R. Pattipati, "A Markov decision problem approach to goal attainment," *IEEE Trans. Syst., Man, Cybern., Part A: Syst. Humans*, vol. 38, no. 1, pp. 116–132, Jan. 2008.
[62] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuska, "Approximate reinforcement learning: An overview," in *Proc. IEEE Symp. Adap. Dyn. Program. Reinforcement Learn.*, 2011, pp. 1–8.
[63] S. A. Jacklin, "Survey of verification and validation techniques for small satellite software development," in *Proc. Space Tech Expo Conf.*, 2015, pp. 1–20.
[64] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robot. Comput.-Integr. Manuf.*, vol. 27, no. 1, pp. 135–149, Feb. 2011.
[65] L. Zuo, Q. Guo, X. Xu, and H. Fu, "A hierarchical path planning approach based on A* and least-squares policy iteration for mobile robots," *J. Neurocomput.*, vol. 170, pp. 257–266, 2015.
[66] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[67] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, 2003.
[68] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: recent advances and applications," *Inf. Sci.*, vol. 261, pp. 1–31, 2014.
[69] C. B. Williams and P. P. Nayak, "A model-based approach to reactive self-configuring systems," in *Proc. 13th Nat. Conf. Artif. Intell.*, 1996, pp. 971–978.
[70] X. Gao, J. Zhang, N. Ning, and J. Xue, "The Livingstone model of a spacecraft power system," in *Proc. Int. Conf. Meas. Technol. Mechatron. Autom.*, 2010, pp. 896–899.
[71] M. Bozzano *et al.*, "On-board autonomy via symbolic model based reasoning," in *Proc. 10th ESA Workshop Adv. Space Technol. Robot. Autom.*, 2008, pp. 1–8.
[72] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory Practice*. San Francisco, CA, USA: Morgan Kaufmann, 2004.
[73] A. Indra, V. K. Agrawal, and V. V. S. Sarma, "Stratified agent architecture for on-board mission planning and execution for an autonomous spacecraft," in *Proc. TENCON 2008, 2008 IEEE Region 10 Conf.*, 2008, pp. 1–6.
[74] E. Vassev, M. Hinchey, and J. Paquet, "Towards an ASSL specification model for NASA swarm-based exploration missions," in *Proc. 2008 ACM Symp. Appl. Comput.*, 2008, pp. 1652–1657.
[75] X. Ruia, C. Pingyuana, and X. Xiaofeib, "Realization of multi-agent planning system for autonomous spacecraft," *Adv. Eng. Softw.*, vol. 36, no. 4, pp. 266–272, 2005.
[76] I. Papusha, J. Fu, U. Topcu, and R. Murray, "Automata theory meets approximate dynamic programming: optimal control with temporal Logic constraints," in *Proc. 55th IEEE Conf. Decis. Control*, 2016, pp. 434–440.
[77] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. 55th IEEE Conf. Decis. Control*, 2016, pp. 6565–6570.

Massimo Tipaldi received the Master's degree in computer science engineering (with a specialization in automation and control systems) from the University of Sannio, Benevento, Italy, working toward the Ph.D. degree focusing on spacecraft autonomy and its implementation via approximate dynamic programming and model checking techniques.

He is a Space Software Project Manager/System Engineer with OHB Italia Spa/OHB System AG, and is currently working for the PLAnetary Transits and Oscillations of stars (PLATO) space observatory. He possesses 18 years of experience in the managerial and technical coordination of ESA/ASI/CNES SW projects (satellite systems, experimental equipment for the International Space Station, and ground segments). He holds 1 patent and has co-authored more than 20 papers published in proceedings of international conferences or international archival journals. His research interests include spacecraft on-board SW reference architecture, SW development process, formal verification, FDIR, and advanced system control techniques.

Luigi Glielmo (S'84–A'85–M'90) was born in 1960. He received the Laurea degree in electronic engineering and Research Doctorate degree in automatic control.

He taught at the University of Palermo, the University of Naples Federico II, and the University of Sannio, Benevento, Italy, where he is currently a Professor of automatic control. From 2001 to 2007, he was the Head of the Department of Engineering, University of Sannio, where he is currently a Rector's Delegate for technology transfer and coordinator of the Ph.D. course on information technologies for engineering. He has co-authored more than 130 papers on international archival journals or proceedings of international conferences, coedited 2 books, and holds 3 patents. His research interests include singular perturbation methods, model predictive control methods, automotive controls, deep brain stimulation modeling and control, and smart-grid control.

Dr. Glielmo has been a member of Editorial Boards of prestigious archival journals and was the Chair of the IEEE Control Systems Society Technical Committee on Automotive Controls.