Agent-based architectures supporting fault-tolerance in small satellites

Carvajal Godínez, J.

DOI
10.4233/uuid:b528d7be-e82d-4205-abdf-3fb3fa7f1011

Publication date
2021

Document Version
Final published version

Citation (APA)
Carvajal Godínez, J. (2021). *Agent-based architectures supporting fault-tolerance in small satellites*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:b528d7be-e82d-4205-abdf-3fb3fa7f1011

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Agent-based Architectures Supporting Fault-Tolerance in Small Satellites

AGENT-BASED ARCHITECTURES SUPPORTING FAULT-TOLERANCE IN SMALL SATELLITES

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Monday 8 February 2021 at 10:00 o'clock

by

Johan CARVAJAL-GODÍNEZ

Master of Engineering in Modern Manufacturing Systems,
Instituto Tecnológico de Costa Rica, Cartago, Costa Rica,
born in San Jose, Costa Rica.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. E.K.A. Gill	Delft University of Technology, promotor
Dr. J. Guo	Delft University of Technology, copromotor

Independent members:

Prof. dr. A. van Deursen	Delft University of Technology
Prof. dr. ir. M. Mulder	Delft University of Technology
Prof. dr. S. Montenegro	Wurzburg University
Prof. dr. E. Caldwell	University of Costa Rica
Dr. Ir. M. Verhoef	European Space Agency

This research was funded by Costa Rica Institute of Technology, and also supported by the Delft University of Technology. Prof. dr. E.K.A. Gill and Dr. Jian Guo contributed significantly to the realization of the thesis.

Keywords: Small Satellites, Onboard Software, Multi-Agent Systems, Satellite Software Architecture, Onboard Satellite Communication, Fault Tolerance

Printed by: IPSKAMP printing

Front & Back: Becki Corrales Brenes

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

Alan Turing

Dedicated to my father Alexis Carvajal-Arias (1946-2019)

CONTENTS

SUMMARY

Since the launch of the first artificial satellite in October 1957, both satellite computers and their onboard software have changed significantly to integrate more functionalities and making mission operations more reliable. As a result, satellites have become more sophisticated and mission designers have moved functionality from the ground segment to the satellite onboard computers to make them more autonomous. In fact, a larger number of satellite components are adopting the use of embedded computers to improve their performance and increasing subsystems miniaturization. This can be seen by the growth in the number of small satellite missions ranging from 1 kg to 100 kg of mass launched during the past 10 years.

Software-defined components increase the complexity of the onboard software. They demand more computing power on the microprocessors onboard, but they also offer several benefits. For instance, software-based components have the capability to reconfigure and adapt to operations environment to allow updating and upgrading the satellite subsystems without changing their physical architecture. That makes satellites more resilient to failures. The flexibility in the software implementation also allows code reuse, which impacts the overall project cost.

One mechanism to deal with the increased systems and software complexity is by enabling novel architectures. These new approaches are intended to mitigate the increment of software complexity, since they have the ability to deal with multiple views of the system including both its functional and non-functional aspects. Agent-based software engineering and Multi-Agent Systems (MAS) technologies are ideal to deal with software complexity, since they provide the theoretical foundations and the tools required to develop reliable distributed software applications. This architectural approach was taken as a baseline to develop this dissertation on Agent-based Architectures supporting Fault-Tolerance in Small Satellites.

This research proposes the adoption of a MAS-based approach for on-board software architecture design with the purpose of enabling mechanisms to handle increased software complexity in time-critical and safety-critical subsystems of satellites. The Attitude Determination and Control Subsystem (ADCS) of small satellite missions is taken as case study for verification and validation of the algorithms, models and methodologies proposed in this dissertation since the ADCS requires high computing and communication capabilities for its implementation.

Firstly, the applicability of MAS-based architectures on Fault Detection Isolation and Recovery (FDIR) algorithms for onboard software of ADCS subsystems was investigated. A review and comparison of FDIR methods was conducted to define a proper approach. Then a design of an agent-based algorithm for detecting and correcting drift on gyroscopes was proposed and implemented at simulation level for the ADCS case study.

In parallel, this research identified a list of services required for MAS-based software implementation that can be applied to satellite subsystems. A qualitative analysis of ex-

isting MAS development frameworks was performed. From this comparison, four key components were identified, namely agents, messages, organizations, and platforms. Based on the interaction of these components two main capabilities for behavior allocation and communication were studied in detail. The main effort was put on the implementation of the communication data bus that enables agents' interactions in a linear bus topology often found on ADCS implementations. For that purpose, an analytical model of bus utilization was derived to quantify the data bus load as a function of its implementation and operation parameters. This model was verified and validated against a discrete time simulation model controlling both implementation and operation parameters for an ADCS case study.

The proposed MAS-based software architecture required to define new methods and tools for its implementation on small satellites. For that reason, a complete chapter was devoted to describe a methodology proposed for that purpose. A comparison of model-driven software development methodologies was performed to identify design gaps, but also to understand how these model-based methods fit into the software development cycle of space systems. Once the gaps were identified, a development methodology was proposed to integrate end-to-end activities for model-driven software development with multi-agent systems for satellites. The proposed methodology was called Multi-Agents Systems for Satellite Applications (MASSA). Quantitative experiments compared the implementation of attitude estimation algorithms using the proposed methodology and compared the numerical results to simulation models for verification and validation.

Finally, in order to optimize the architectural implementation of the proposed MAS-based onboard software it was found that the organization of agents, particularly its organizational topology was key for supporting fault-tolerance in small satellites. Following a qualitative analysis and including front-end systems engineering aspects, an objective function for optimizing the cost of communication for agents' organization was established and solved using a Genetic Algorithms (GA) approach. The implementation provides a characterization of the proposed GA algorithm to show performance aspects such as time to find a solution and number of generations required to find a feasible solution so that designers can improve and speed up their software development process. This dissertation provided the following contributions to the existing body of knowledge: First, a novel Fault Detection Isolation and Recovery architecture combining model-based and data-driven techniques to detect and correct sensor errors on ADCS. Second, a new bus utilization model for distributed data communication buses that can be used to balance out the performance of MAS-based software implementations. Third, an innovative model-driven methodology for designing fault-tolerant MAS-based software. Fourth, a software library for agent-based software development in miniaturized satellites, and fifth, a genetic algorithm for optimal organization of agent-based software.

This thesis focused on addressing the software complexity problem through the use of agent-based software architectures. That was achieved by extending the state-of-art component-based design implemented with object-oriented paradigms to include multi-agent systems concepts. Future space missions with small satellites can benefit from these contributions to boost their performance and enable autonomous fault detection, isolation, and recovery. Sergei Korolev once said "I believe in the future. It is wonderful because it stands on what has been achieved.", so it was my thesis.

SAMENVATTING

Sinds de eerste lancering van een kunstmatige satelliet in oktober 1957, zijn satellietcomputers en hun ingebouwde software aanzienlijk veranderd om meer functionaliteiten te integreren en missieoperaties betrouwbaarder te maken. Als gevolg hiervan zijn satellieten geavanceerder geworden en hebben missieontwerpers functionaliteit op de grond overgedragen naar de boordcomputers van satellieten om ze autonomer te maken. Tegenwoordig maakt een steeds groter aantal satellietcomponenten gebruik van ingebouwde computers om hun prestaties te verbeteren en de miniaturisatie van subsystemen te vergroten. Dit is te zien aan de groei van het aantal kleine satellietmissies, variërend van 1 kg tot 100 kg aan massa, die in de afgelopen 10 jaar gelanceerd zijn.

Software aan boord is complexer vanwege de softwaregedefinieerde componenten. Zij vragen meer rekenkracht van de microprocessors aan boord, maar bieden ook verschillende voordelen. Deze componenten hebben bijvoorbeeld de mogelijkheid om opnieuw geconfigureerd te worden aan de operationele omgeving om het bijwerken en upgraden van de satellietsubsystemen mogelijk te maken zonder hun fysieke architectuur te wijzigen. Dat maakt satellieten beter bestand tegen storingen . De flexibiliteit in de software-implementatie maakt ook hergebruik van code mogelijk, wat voordelige gevolgen heeft voor de totale projectkosten.

Een mechanisme om met de toegenomen complexiteit van systemen en software om te gaan is door nieuwe architecturen mogelijk te maken. Deze nieuwe benaderingen zijn bedoeld om de toename van softwarecomplexiteit te verminderen, aangezien zij in staat zijn om met meerdere weergaven van het systeem om te gaan, zowel de functionele als de niet-functionele aspecten ervan. Agent-gebaseerde software engineering en Multi-Agent Systems (MAS) -technologieën zijn ideaal om met softwarecomplexiteit om te gaan, aangezien zij de theoretische basis en de tools bieden die nodig zijn om betrouwbare gedistribueerde softwareapplicaties te ontwikkelen. Deze architecturale benadering werd als uitgangspunt genomen voor de ontwikkeling van dit proefschrift over agent-gebaseerde architecturen die fouttolerantie in kleine satellieten ondersteunen .

Dit onderzoek stelt de toepassing van een MAS-gebaseerde benadering voor het ontwerpen van software-architectuur aan boord met als doel mechanismen in staat te stellen om de toegenomen softwarecomplexiteit in tijds- en veiligheidskritische subsystemen van satellieten aan te pakken. Het “Attitude Determination and Control Subsystem” (ADCS) van kleine satellietmissies wordt beschouwd als case study voor verificatie en validatie van de algoritmen, modellen en methodologieën die in dit proefschrift worden voorgesteld, aangezien de ADCS hoge reken- en communicatiemogelijkheden vereist voor de implementatie ervan.

Ten eerste werd de toepasbaarheid van op MAS gebaseerde architecturen op “Fault Detection Isolation and Recovery” (FDIR) -algoritmen voor onboard-software van ADCS-subsystemen onderzocht. Er werd een evaluatie en vergelijking van de FDIR-methoden uitgevoerd om een juiste aanpak te definiëren. Vervolgens werd een ontwerp van een

agent-gebaseerd algoritme voor het detecteren en corrigeren van drift op gyroscopen voorgesteld en geïmplementeerd op simulatieniveau voor de ADCS case study.

Tegelijkertijd werd in dit onderzoek een lijst met services geïdentificeerd die nodig zijn voor MAS-gebaseerde software-implementatie die kan worden toegepast op satellietsubsystemen. Er is een kwalitatieve analyse van bestaande MAS-ontwikkelingskaders uitgevoerd. Uit deze vergelijking, werden vier belangrijke componenten geïdentificeerd, namelijk agenten, berichten, organisaties en platforms. Op basis van de interactie van deze componenten werden twee hoofdcapaciteiten voor gedragstoewijzing en communicatie in detail bestudeerd. Het meeste nadruk werd gelegd op onderzoeken van de implementatie van de communicatiedatabus die interacties van agenten mogelijk maakt in een lineaire bustopologie die vaak wordt aangetroffen in ADCS-implementaties. Voor dat doel werd een analytisch model van busgebruik afgeleid om de databusbelasting te kwantificeren als functie van de implementatie- en operationele parameters. Dit model werd geverifieerd en gevalideerd met behulp van een discreet tijdssimulatiemodel dat zowel de implementatie- als de operationele parameters bestuurt voor een ADCS-casestudy.

De voorgestelde MAS-gebaseerde softwarearchitectuur vraagt om nieuwe methoden en hulpmiddelen te definiëren voor de implementatie ervan op kleine satellieten. Om die reden is er een volledig hoofdstuk gewijd aan het beschrijven van een methodologie die voor dat doel is voorgesteld. Een vergelijking van model-gestuurde softwareontwikkelingsmethodologieën werd uitgevoerd om gaten in het ontwerp te identificeren, maar ook om te begrijpen hoe deze, op modellen gebaseerde methoden, passen in de softwareontwikkelingscyclus van ruimtesystemen. Toen de hiaten eenmaal waren vastgesteld, was er een ontwikkelingsmethode voorgesteld om " end-to-end-activiteiten" voor modelgestuurde softwareontwikkeling te integreren met multi-agentsystemen voor satellieten. De voorgestelde methodologie heet "Multi-Agents Systems for Satellite Applications" (MASSA). Kwantitatieve experimenten vergeleken de implementatie van attitudeschattingsalgoritmen met behulp van de voorgestelde methodologie en vergeleken de numerieke resultaten met simulatiemodellen voor verificatie en validatie.

Tenslotte, om de architectonische implementatie van de voorgestelde MAS gebaseerde software aan boord te optimaliseren, werd ontdekt dat de organisatie van agenten, met name de organisatorische topologie, cruciaal was voor het verbeteren van fouttolerantie in kleine satellieten. In aansluiting op een kwalitatieve analyse en inclusief aspecten van front-end systems engineering, werd een objectieve functie voor het optimaliseren van de communicatiekosten voor de organisatie van agenten opgesteld en opgelost met behulp van een Genetic Algorithms (GA) -benadering. De implementatie biedt een karakterisering van het voorgestelde GA-algoritme om prestatieaspecten te laten zien, zoals de tijd om een oplossing te vinden en het aantal iteraties dat nodig is om een haalbare oplossing te vinden, zodat ontwerpers hun softwareontwikkelingsproces kunnen verbeteren en versnellen . Dit proefschrift leverde de volgende bijdragen aan de bestaande kennis: ten eerste, een nieuwe Fault Detection Isolation and Recovery-architectuur die modelgebaseerde en datagestuurde technieken combineert om sensorfouten op ADCS te detecteren en corrigeren. Ten tweede, een nieuw busgebruiksmodel voor gedistribueerde datacommunicatiebussen dat kan worden gebruikt om de prestaties van op MAS gebaseerde software-implementaties te compenseren. Ten derde, een

innovatieve model-gestuurde methodologie voor het ontwerpen van fouttolerante MAS-gebaseerde software. Ten vierde, een softwarebibliotheek voor agent-gebaseerde softwareontwikkeling in geminiaturiseerde satellieten, en ten vijfde, een genetisch algoritme voor een optimale organisatie van agent-gebaseerde software.

Dit proefschrift richtte zich op het aanpakken van het softwarecomplexiteitsprobleem door het gebruik van agentgebaseerde softwarearchitecturen. Dat werd bereikt door het geavanceerde, op componenten gebaseerde, ontwerp dat geïmplementeerd werd met objectgeoriënteerde paradigma's, uit te breiden met multi-agent-systeemconcepten. Toekomstige ruimtemissies met kleine satellieten kunnen profiteren van deze bijdragen om hun prestaties te verbeteren en autonome foutdetectie, isolatie en herstel mogelijk te maken. Sergei Korolev zei ooit: "Ik geloof in de toekomst. Het is geweldig omdat het staat op wat er is bereikt.", zo was mijn proefschrift.

ACRONYMS

ACC Agent Communication Channel. 12, 13, 65, 68–70

ACL Agent Communication Language. 64–71

ADCS Attitude Determination and Control Subsystem. xi, xii, 18–23, 28, 36, 55, 63, 78–82, 85, 88, 89, 103, 105–108, 110–113, 117, 118, 142, 143

AGC Apollo Guidance Computer. 3

AI Artificial Intelligence. 2

AIP Agent Interaction Protocols. 64, 67, 68, 143

AMFT Adaptive Middleware for Fault-Tolerance. 7

AMS Agent Management System. 12, 13, 120, 121, 129, 130, 133, 135

ANN Artificial Neural Networks. 42

AOCS Attitude and Orbit Control Subsystem. 8, 10, 15, 20, 25–27, 33, 34, 40, 43, 44, 46, 47, 51–53, 56, 60, 64, 68, 71, 72, 74, 75, 77, 91, 92, 126–132, 135–137, 140

AP Agent Platform. 13

AUML Agent Unified Modeling Language. 12

BFA Brute Force Algorithm. 122, 123, 139, 140, 147

BU Bus Utilization. 74, 75, 82–84, 92

CAN Controller Area Network. 63, 66, 68–70, 72, 75–84, 91, 92, 144, 146

CCM Cost of Communication Matrix. 116

CDHS Command and Data Handling Subsystem. 53, 80, 92, 103, 105, 108

CoM Center of Mass. 28

COMPASS Correctness, Modeling and Performance of Aerospace Systems. 108

CORBA Common Object Request Broker. 12, 66

COTS Commercial-Off-The-Shelf. 4, 6–8, 13, 18, 58

CPS Cyber-Physical System. 26, 27

CPU Central Processing Unit. 6

CRF Controller Reference Frame. 28, 29, 31, 36, 37

DCM Direction Cosine Matrix. 29, 31

DF Directory Facilitator. 12, 13

DSL Domain Specific Language. 98

DTW Dynamic Time Warping. 41

ECEF Earth-Centered Earth-Fixed. 27–29

ECI Earth-Centered Inertial. 27–31, 36, 37

EKF Extended Kalman Filter. 36–38, 72, 80, 103, 106–112, 173

EPS Electric Power Subsystem. 15

ESA European Space Agency. 14, 126, 146

FDI Fault Detection and Identification. 43–46, 51, 53, 55, 60, 61, 131, 142

FDIR Fault Detection Isolation and Recovery. xi, xii, 3, 7, 8, 10, 17–22, 40–47, 49–53, 60, 61, 69, 94, 97, 98, 102, 106, 109–112, 118, 126, 128, 129, 131, 137, 142–146

FIPA Foundations of Intelligent Physical Agents. 12, 17, 64–69, 92

FIR Fault Isolation and Recovery. 43–46, 52–54, 56, 59–61, 131, 132, 137, 142

FPGA Field Programmable Gate Arrays. 2, 6

FTC Fault-Tolerant Control. 40, 44

GA Genetic Algorithms. xii, 119, 123–126, 128, 129, 131–140, 147

GDC Gemini Digital Computer. 2

GNSS Global Navigation Satellite System. 34

GPS Global Positioning System. 2, 34, 71, 103, 104, 127, 130

GPU Graphic Processing Unit. 2

HTTP Hypertext Transfer Protocol. 12, 66, 68

IAT Inter-Arrival Time. 74

IBM International Business Machines. 2, 3

ICA Independent Component Analysis. 42, 45, 52, 53, 57, 59–61

IIOP Internet Inter-Orb Protocol. 10, 12, 66, 68

IMU Inertial Measurement Units. 2, 50–52, 80, 103

IPMT Internal Platform Message Transport. 12, 65, 69, 70

JADE Java Agent Development Framework. 12–14, 68, 110

JVM Java Virtual Machine. 13

KQML Knowledge Query Manipulation Language. 66

LEO Lower Earth Orbit. 6, 27, 28, 33, 34

MAC Medium Access Control. 75

MAS Multi-Agent Systems. xi, xii, 11–22, 25–27, 44, 45, 64, 68–70, 79, 92, 94–98, 105–108, 110, 111, 113, 116–122, 125, 127, 132, 134–138, 140, 142–145

MASSA Multi-Agents Systems for Satellite Applications. xii, 23, 97–100, 102–106, 108–113, 146

MDE Model-Driven Engineering. 17, 94

MTP Message Transport Protocol. 64, 67, 68, 92

MTS Message Transport System. 65, 68

NASA National Aeronautics and Space Administration. 8, 14, 58

NORAD North American Aerospace Defense Command. 171

OBC On-Board Computer. 6, 7, 22, 79, 81, 99, 107, 120, 129

OBSW On-Board Software. 7, 14, 18, 19, 23, 43–45, 95, 99, 105, 112, 118, 140, 142, 145

OOA Object-Oriented Architecture. 19

OOL Object-Oriented Language. 13

ORF Orbital Reference Frame. 28, 29

OSI Open Systems Interconnection. 68

PCA Principal Component Analysis. 42

PCB Printed Board Circuit. 5

PDO Process Data Object. 68–71, 79, 92

PLS Partial Least Squares. 42

PROBA Project for On-Board Autonomy. 14, 40, 126–130, 132, 135–137, 140

PSO Particle Swarm Optimization. 119, 123–125, 140, 147

RISC Reduced Instruction Set Computer. 3

RMI Remote Method Invocation. 10, 12, 66, 68

RTOS Real-Time Operating System. 69, 97, 110

SBRF Satellite Body-Fixed Reference Frame. 28–31, 37

SDO Service Data Objects. 70

SDR Software-Defined Radio. 6

SEU Single-Event Upset. 3, 4

SLIM System-Level Integrated Modeling. 108

SOC System-On-Chip. 6

SPADE Smart Python Development Environment. 68

SPARC Scalable Processor Architecture. 3

SPE Squared Prediction Error. 51

SSTM SysML to SLIM Transformation Methodology. 108

SVM Support Vector Machine. 42

SysML Systems Modeling Language. 98–100, 102, 105, 108, 146

TCP Transmission Control Protocol. 66

TLE Two-Line Elements. 171

TRL Technology Readiness Level. 6, 69, 126

TT&C Telemetry, Tracking and Control. 15

UART Universal Asynchronous Receiver-Transmitter. 110

UDP User Datagram Protocol. 66

UML Unified Modeling Language. 98

WAP Wireless Application Protocol. 68

XML eXtensible Markup Language. 66

XMPP Extensible Messaging and Presence Protocol. 12, 68

List of Symbols

Ω_a Right ascension of the ascending node. 28

μ_E Gravitational coefficient of the Earth. 28

ν_a True Anomaly at epoch t_0. 28

ω_p Argument of the perigee. 28

θ Transformation angle from reference system A to reference system B. 30

$\mathbf{A}(\mathbf{q})$ Rotation matrix given a quaternion. 30

$\mathbf{I}_{sat}$ Inertia matrix of the satellite. 31

$\mathbf{L}$ Angular Momentum. 31

$\mathbf{N}_{ctrl}$ Control torque. 31

$\mathbf{N}_{dis}$ Disturbance torque. 31

$\mathbf{N}_{ext}$ External torque. 31

$\mathbf{S}(\cdot)$ 3x3 Symmetric skew matrix. 31

$\mathbf{a}_d$ Perturbing acceleration of a satellite. 28

$\mathbf{b}_{gyro}$ Measurement bias for gyroscope. 32

$\mathbf{b}_{mag}$ Measurement bias for magnetometer. 32

$\mathbf{e}$ Attitude parameter. 29

$\mathbf{q}$ Spacecraft attitude quaternion. 28, 32

$\mathbf{r}$ Geocentric position vector. 28

$\mathbf{v}_{mag}$ Magnetic field vector measurement. 37

$\mathbf{v}_{sun}$ Sun vector measurement. 37

$\mathbf{\Omega}(\cdot)$ 4x4 skew symmetric matrix. 30

$\boldsymbol{\omega}$ Angular velocity vector. 30, 32

a Semi-major axis of an ellipse. 28

e Eccentricity of an elliptical orbit. 28

i_o Inclination of an orbit plane. 28

s Quaternion scalar. 29

t_e Reference time of epoch. 28

t Time. 28

1

INTRODUCTION

The computer is the most remarkable tool that we've ever come up with.
It's the equivalent of a bicycle for our minds.

Steve Jobs

Abstract

Developing more capable satellite missions faces increased onboard software complexity. The next generations of satellites need to enable, among others, the infrastructure for intelligent algorithm execution, as well as the capabilities for proactive failure detection, isolation, and recovery. Agent-based architectures are a new developing approach adopted in software engineering that combines flexibility, scalability and adaptability to dynamic operating environments while providing a framework to implement state-of-the-art algorithms with artificial intelligence. This Chapter focuses on describing the motivation for this dissertation based on the evolution of spacecraft computers, as well as the trends in small satellite engineering. It identifies the link between the increase in onboard software features and onboard software complexity. To address that issue, this research proposes the adoption of multi-agent-based software architectures as an extension of the well-known object-oriented software architecture. The main goal is to make satellite missions more resilient to failures during operations. It also describes the infrastructure needed for the development and implementation of multi-agent based software architectures. Finally, it introduces the research motivation, questions and methodology, and it describes the research structure used to develop this dissertation.

1

Computers are the core of any electronic device nowadays. Satellites are not the exception. In fact, an increasing number of satellite components are adopting the use of embedded computers to improve their performance and increasing subsystems miniaturization. Examples of such components include star trackers, Global Positioning System (GPS) receivers, and Inertial Measurement Units (IMU). For that reason, the satellite's onboard software design process shall evolve to embrace changes in processors implementations, as well as new computing paradigms that exploit concurrent processing technologies, for instance hardware accelerators like Graphic Processing Unit (GPU), Field Programmable Gate Arrays (FPGA), and Artificial Intelligence (AI) enabled microprocessors.

1.1. THE EVOLUTION OF SPACECRAFT COMPUTERS

Since the launch of the first artificial satellite in October 1957, both computers and their software have changed significantly to integrate more functionalities and making mission operations more reliable. In parallel, the advances of the silicon integrated microprocessors in the 70's and several other software innovations, such as compilers, have paved the way for more advanced applications onboard of space systems. However, as discussed by Eickhoff (2011), it was the Apollo Program that raised the need to incorporate electronically controlled mechanisms for life support, trajectory control, landing control, Moon re-launch, docking/undocking control and re-entry capabilities on spacecraft.

The first computer to fly to outer space was manufactured by IBM using discrete transistors. It was the Gemini Digital Computer (GDC), which consisted of a customized 39-bit processor architecture with 4K word storage capacity implemented using magnetic core memory. It operated at a clock frequency of 7 kHz, and it took up to 840 ms to perform the most advanced mathematical operations in the moment, divisions. The mass of this computer was 27 kg, and its power consumption was about 100 W. According to Cooper and Chow (1976), later developments during the 60's and early 70's improved software support capabilities for assembly language programming and functional test features. Figure 1.1 illustrates the layout of a Gemini Digital Computer.

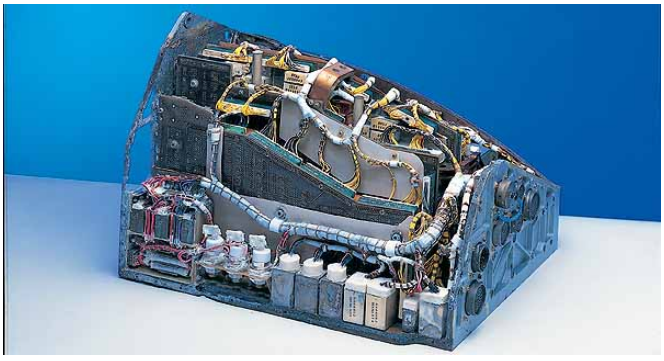


Figure 1.1: Gemini Digital Computer (Source: Virtual AGC Project)

During the 70's the Apollo program incorporated integrated circuit technology to reduce its Apollo Guidance Computer (AGC) mass and increasing its performance. One of the biggest advancements in the software of the AGC was the integration of a real time multitasking sub-layer to support up to 8 parallel tasks. Even when the clock speed was 1 MHz, the AGC struggled to cope with lunar approaches maneuvers as described by Tomayko (1985).

The Space Shuttle program increased the reliability of space computers by enabling hardware configurations for voting. Their microprocessors were inspired by IBM 360 architecture and they were adapted for radiation tolerance. From a software development perspective, these computers supported a higher abstraction assembly language, that reduced the development cycle and allowed higher separation of concerns. Also, the onboard software added support for linear algebra operations required mainly for guidance and navigation control as discussed by Klinar et al. (1975).

Later, satellites and space probes required more advanced onboard computing capabilities delivered in the x86 and other microprocessor architectures. For example, the Voyager probes needed to split up command and data handling from attitude control, as well as payload functionalities. That was the beginning of distributed spacecraft architectures. In parallel, software features like built-in self-test were incorporated for Fault Detection Isolation and Recovery (FDIR) as discussed in the work by Rasmussen and Litty (1981). Additionally, the onboard software required features for performance optimization, for example, trajectory control. More complexity in these missions led to having more sophisticated ways to deal with integration. For that purpose, Gangl (2013) argues the need to have standards to make the interface management easier. Standards like MIL-STD-1750 and MIL-STD-1815 described the requirements for chip manufacturers to design compatible devices for different mission requirements.

The adoption of microprocessor architectures based on a Reduced Instruction Set Computer (RISC), especially, the Scalable Processor Architecture (SPARC) in the LEON processor have enabled the adoption of more advanced real-time operating systems and compilers, which enable capabilities to develop more complex software for satellite missions. However, much more has to be done to satisfy the increased demand for onboard computing power to support intelligent applications. Sarode and Patil (2016) describe how the LEON3FT processor implements a SPARC processing core with a floating point unit and a memory controllers to support safety critical space missions with native fault tolerant features. Figure 1.2 shows a LEON-Express Single-Event Upset (SEU) test board to illustrate packaging and interfaces for debugging.

1.2. TRENDS IN MINIATURIZED SATELLITES ENGINEERING

In general, satellites are classified according their mass. In Barnhart et al. (2009a) the term nano-satellite is used for spacecraft with a mass in the range of 1 to 10 kg, while micro-satellite describes a satellite with a mass in the range of 10 to 100 kg.

According to a recent nano/micro satellite market research presented by Doncaster et al. (2016), the launch of satellites in the range of 1 to 50 kg will increase about 10% every year between 2016 and 2020. The emerging of a miniaturized spacecraft market offers opportunities for start-ups and well-established companies and organizations dedicated to the engineering of small satellites, mainly those focused on earth observation

Figure 1.2: LEON-Express SEU test board (Source: Gaisler (2002))

services, for example, Planet or broadband internet services such as SpaceX Starlink.

New spacecraft configurations for formation flying and satellite constellations are being demonstrated with nanosatellites in the upcoming years as described by Gill et al. (2013) and Guo and Gill (2013). Three major trends have been identified in the development cycle of small satellite projects:

1. The adoption of Commercial-Off-The-Shelf (COTS) components in the developing process, as discussed by Underwood et al. (1998), which has reduced the financial cost of satellite projects.

2. The adoption of CubeSat standard for launching and deploying satellites, especially in nanosatellites as discussed by Heidt et al. (2000). According to Hubbard (2014) the return on investment of small satellite missions based on CubeSats have increased in the last five years due to a higher availability of launch opportunities.

3. The adoption of miniaturized distributed space systems, which has enabled a wide range of new space applications as discussed in the work of Barnhart et al. (2007).

Satellite missions have become more sophisticated due to the evolution of computing hardware thanks to Moore's law as discussed by Keyes (2006). Designers have moved functionality from the ground segment to the satellite's onboard computers to make them more autonomous as presented by Macdonald and Badescu (2014).

The increase in the small satellites market combined with availability of new computing technologies motivates the focus on small satellites in this dissertation. However, the knowledge derived from it can be used on bigger satellites as well. The following subsection elaborates on current satellite trends, and it discusses the challenges associated with small satellite development. This was used as the baseline to begin the research presented in this thesis.

1.2.1. SUBSYSTEM MINIATURIZATION

The standardization of the mechanical deployment interface has been one of the drivers for the increase in the number of small satellite launches. Puig-Suari et al. (2001) argues that the release of the CubeSat standard in 2004 allowed a reduction in the development time and cost of satellites. It also offered a standard that can be shared by a large number of launch providers. It was initially intended for university development projects, but it was adopted later by industry. Hitt et al. (2016) describe how Cubesats are currently being supported and promoted by national space agencies as a mean for improving their access to space.

Constraints on the mass, volume, and power budgets have required satellite developers to come up with disruptive innovations to fulfill the needs of their missions. For example, Barnhart et al. (2009b) proposed the PCBSat, a micro-engineered space system aiming to provide low cost and high quantities for distributed space missions. In such missions, the integration of several subsystems into highly dense Printed Board Circuit (PCB) is an enabler for the mission success.

Subsystems like propulsion and communication are being re-engineered to enable more compact and capable satellites. In propulsion, for example, new devices called micro-thrusters proposed in the work of Guerrieri et al. (2016) are being designed and tested to ensure they can provide the required mission performance. In communication, new antennas and transceivers designed by Gao et al. (2009) are intended to improve performance, but also to reduce risk during integration, deployment and early satellite's operations. From the onboard computer perspective, Speretta et al. (2016) describes the latest developments being carried out in Pocket Qubes to enable more capable computing devices into smaller form factors. Figure 1.3 shows an onboard computer from Alba Orbital for a Pocket Qube with 42 mm x 42 mm dimensions and a mass of about 10 grams.

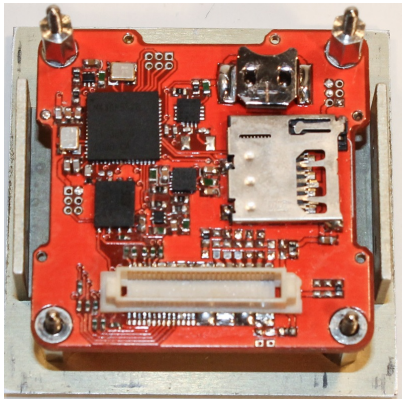


Figure 1.3: Pocket Qube onboard computer (Source: Alba Orbital)

One trend in the miniaturization of onboard computers is the adoption of highly integrated systems for flexible software implementation allowing the migration from hardware to software-defined components.

1.2.2. SOFTWARE-DEFINED COMPONENTS

The high availability of System-On-Chip (SOC) and FPGA-based technology have supported the evolution from hard-wired capabilities to more flexible features in highly integrated systems. One example on satellites is the implementation of Software-Defined Radio (SDR) for both the space and ground segment. Besides flexibility in the implementation, SDR communication platforms enable an additional reduction in cost and mass, which is one of the main drivers in miniaturized satellite applications as discussed by Maheshwarappa et al. (2015).

Software-defined components increase the complexity of the onboard software. They also demand more computing power on the microprocessors onboard, but they also bring several benefits. One of the main advantages is the adaptive and reconfigurable capabilities that allow updating and upgrading the satellite subsystems without changing their physical architecture. The flexibility in the software implementation also allows the code reuse, which impacts the overall project cost.

1.2.3. EMERGING ONBOARD COMPUTING TECHNOLOGIES

Satellite's OBC are heavily influenced by consumer electronics, in particular by the mobile phone industry. They share some common requirements and constraints, for instance, reduced volume, mass and power, and increasingly demanding applications. Phones manufacturers have come up with design innovations in the microprocessor to cope with these demands. There is an opportunity of adopting them in the aerospace applications.

For satellite applications, there are some restrictions in adopting this current computing technologies. For example, multicore processors are not yet mature enough for satellite applications besides Lower Earth Orbit (LEO) since the reliability requirements for radiation tolerance requires higher Technology Readiness Level (TRL). Also adopting COTS middlewares and machine learning technologies requires effort for its qualification for space use. The following paragraphs describe the potential of these emerging technologies.

Multi-core processors: Concerning multi-core in flight software, two approaches are being explored. The first one is called asymmetric multiprocessing, where each CPU core executes an instance of the operating system. The implementation of asymmetric multiprocessing requires standard inter-process communications for sharing information between cores. In this approach cores are isolated logically and physically from each other, which makes the identification of failures easier. The second approach is the symmetric multiprocessing that allows dynamic task allocation to any core by running a single operating system instance. Both methods are currently developed and supported for the space-rated LEON4 processor, as discussed in Cederman et al. (2014). It is important also to consider that there is a trend on having multi-core enabled computers on satellites that features heterogeneous processing units with CPU and FPGA components.

Distributed Data Buses: The process of distributing functionalities among components creates data networks that require communication protocols to standardize their interface. Traditionally, space systems have their space-qualified communication protocols that consume more energy due to radiation hardened components. In recent days,

the space community is trying to adopt and qualify communication protocols from automobile industry to improve performance and energy efficiency. For instance, Scholz et al. (2017) presents an open source implementation of CAN bus protocol for CubeSats, where they demonstrate that wide used COTS protocols are an option for next generation spacecraft buses.

Distributed data buses are a key enabler for distributed computing models onboard of satellite missions. It is important to research how current FDIR capabilities can be improved as well as enabling the support for concurrent OBSW execution.

Middlewares: One of the mechanisms to enable onboard computing flexibility and code reuse is the adoption of software middleware that handles the interaction between missions applications and the execution infrastructure in the spacecraft bus. Middlewares are adopted as a response to the increase in the software complexity, that is described later on. The middleware provides an abstraction layer to support highly complex and heterogeneous execution components for applications to interact with the hardware components in the OBC.

In aerospace applications, middlewares are used to enable highly concurrent and distributed onboard software. For example, in Fayyaz et al. (2012) an Adaptive Middleware for Fault-Tolerance (AMFT) performs FDIR activities. It also synchronizes the operation of distributed OBC configurations. Other examples of middleware are the ones provided by open-source projects like $Arduino^{TM}$, that enable faster software development and testing as presented in Le Vinh et al. (2015). In summary, middlewares are the mechanism to implement intelligent software layers in the onboard computers of satellites.

Machine Learning: Machine Learning is the product of the use of artificial intelligence techniques and advances in process modeling, as well as data-based engineering. As result of machine learning algorithms, there are more accurate classification and prediction tools that can be implemented onboard satellites, as they are being adopted in autonomous vehicles to learn from their environment and adapting their behavior accordingly. Recent applications for machine learning in OBC engineering include spacecraft autonomy by D'Angelo et al. (2017), and on-orbit sensor calibration for the star trackers' measurement model parameters as presented in Li et al. (2017a).

Hybrid Computing: One of the biggest challenges in OBC design for upcoming space missions is balancing out the performance and energy efficiency that COTS components deliver with the reliability features of radiation-hardened devices. This is what George and Wilson (2018) define as a hybrid OBC architecture. It is important to make the remark that in this case hybrid computers are different from heterogeneous processing units mentioned above.

According to Furano and Menicucci (2018), to avoid single-point failures of space system, all functions executed by a hybrid OBC are required to be internally redundant. That requires establishing interfaces and mechanisms to migrate functions among different processing elements within the OBC of the satellite. The increase of interaction and interfaces produces more complex configurations at hardware, but specially at software level.

1.2.4. INTEGRATED FAULT DETECTION, ISOLATION AND RECOVERY

As mentioned above, one of the most valuable applications taking advantage of advanced computing models onboard satellites is the implementation of integrated Fault Detection Isolation and Recovery (FDIR) features. In fact, this dissertation takes FDIR as one of the key enablers for future space missions, since it allows autonomous operations required by large and distributed space systems.

According to Bittner et al. (2014a), there is no FDIR development process for aerospace coherently addressing the full FDIR lifecycle, which limits the possibility to effectively determine the propagation and impact of failures in time. This can be critical in the achievement of the mission's objectives, but also an opportunity to provide a framework that addresses that problem taking advantage of software techniques that boost the development and implementation of satellite systems. This dissertation focuses on failure detection and isolation rather than recovery, since recovery techniques may vary from mission to mission. It also focuses on enabling the infrastructure required for fault detection and isolation modeling and implementation using a distributed approach, trying to mimic what organic systems do. Finally, FDIR methods are studied on time-critical subsystems, especially those requiring higher processing capabilities, for instance on the Attitude and Orbit Control Subsystem (AOCS).

1.2.5. THE NEED FOR A MORE RELIABLE AND PRECISE AOCS

Autonomous rendezvous and docking has been one of the reasons to have a precise AOCS on a spacecraft. However, recent applications of small satellites such as the broadband service proposed by SpaceX Starlink requires very precise pointing capabilities, as well as performing periodic orbit correction.

Due to the large scale of such systems, it is imperative to have also the implementation of autonomous Fault Detection Isolation and Recovery capabilities to reduce the operation's cost. That is also a motivation for this dissertation to take the Attitude and Orbit Control Subsystem of satellites as case studies for demonstration purposes.

Currently, there is also a trend in enabling laser communication on small satellites. For instance, NASA selected the Aerospace Corporation to develop a technology demonstration mission to test COTS components in optical communications with CubeSats. The proposed concept included an optical payload using beam spreads in the range of miliradians. According to Janson and Welle (2014b), the mission's objective was to establish a communication link between the AeroCube-OCSD satellite and a telescope located in California, USA. This serves as example of several missions developed to test such capabilities.

From the systems perspective trends on satellite's subsystems miniaturization, the evolution of software-defined components and the emerging of onboard computing technologies enable novel FDIR applications on time-critical subsystems such as AOCS to improve the capability of satellites for autonomous operations. The challenge is how to define and describe a proper system and software architecture that handles the increased software complexity derived from enabling newer and better onboard computing functions such as FDIR. For that reason, studying and dealing with the software complexity problem is key on the route of enabling more autonomous satellite systems.

1.3. THE SOFTWARE COMPLEXITY PROBLEM

Banker et al. (1993) describes software complexity either as a measure of the required resources for interaction between software components, as well as the difficulty of understanding of the entire structure and organization of a program. A good metric of software complexity is the relationship between the number of demanded features and the number of lines of code required for their implementation. This was demonstrated empirically by Basili and Perricone (1984) in their work. In this dissertation's context, software complexity is defined as the increase in the levels of interdependence of software components within the implementation of a set of system features distributed across multiple computers onboard the satellite.

It is logical that by constraining the available execution resources, the software complexity increases, as the functionality density does. Then, according to Atkinson and Kühne (2008), two different types of complexities can be identified; the inherent software complexity which is due to the need for providing more features, and the accidental complexity that is the increase in the interdependence of components to make the system more robust. From a systems engineering perspective, inherent complexity is linked to capabilities, whereas accidental complexity is related to systems characteristics. For instance, accidental complexity is affected by the openness of the system, number of software components, among other elements in the systems design.

In software systems, the level of complexity is determined by the interaction between the multiple software components and their environment. Magee and Kramer (1996) describe this interaction in terms of the provided and required services by each software component. Due to the dynamic nature of these interactions, there are emergent behaviors that affect the whole performance of the system, meaning that global performance is more than just adding up the individual components, but requires including behaviors produced by their interface. The effect of these interactions evolve over time so that the future state of the system is affected by the present state. That allows using models, for instance, Markov Chains to describe software systems including its complexity as using techniques such as in the work of Whittaker and Poore (1993) and Béounes et al. (1993).

Software complexity for aerospace applications was introduced and discussed by Vassev and Hinchey (2014) and Lalanda et al. (2013) using the concept of autonomic computing. Since the on-board software is tightly coupled with the hardware components of the spacecraft bus there is a direct link between software complexity and systems size. Yushtein et al. (2011) argues that an increase in the system's complexity makes the link between the avionics and its software more difficult to model due to newer interactions and interfaces. Due to this increase of systems and software complexity:

1. The onboard software requirements definition and design time is increased.
2. The software interface management increases, impacting the verification and validation readiness.
3. The time spent in software development activities becomes shorter since more time is required in front-end systems engineering activities.

The aspects listed above assume that the time for developing the software of a system is fixed by the program management.

A comprehensive analysis of flight software complexity is presented by Dvorak and Lyu (2009), which identified and described 11 recommendations on how to deal with increasing software complexity in on-board software for spacecraft. The main recommendation appoints to the need of improving software architecture design in early stages of spacecraft system development. The study also suggest defining a reference architecture to provide a common list of capabilities required in most of the missions, for instance, navigation, attitude control, thermal control, command and data handling, and FDIR.

1.4. SOFTWARE ARCHITECTURE PARADIGMS

Software architecture is used as a tool to mitigate the increment of software complexity, since it has the ability to deal with multiple views of the system including both its functional and non-functional aspects. According to Mens et al. (2010) to mitigate the increment of software complexity in critical systems, its architecture shall be able to describe, maintain and evolve to adapt without sacrificing its purpose. The architecture shall also enable and facilitating the implementation of software by taking advantage of specific computational models and programming language's features.

There are three main software architecture paradigms that were considered during this research. These are the resource-oriented, service-oriented and object-oriented paradigms. The resource-oriented paradigm involves retrieving information instances from different systems components. Then it performs operation and restores them to the place where they were initially. It can also have multiples copies of the same data distributed among the system's components, which is a concern in highly constrained systems such as satellites.

The service-oriented paradigm involves stateless communication between components using message passing methods. According to Papazoglou (2003), there are endpoint services that implement the data processing and respond to clients when required. This approach relies on describing the system as a collection of services. The main drawback of this architecture paradigm is its dependency in the operations manager to monitor the correctness and overall functionality of services.

Wang and Fung (2004) describe the object-oriented architecture as a collection of objects and classes that have state, behavior and identity. The communication among the objects is stateful and optimized to minimize the workload in the network. One of the advantages of this paradigm is its flexibility to adapt behaviors due to polymorphism. The main drawback of this paradigm is its non-friendly protocols for communication, for instance Internet Inter-Orb Protocol (IIOP) and Java-Remote Method Invocation (RMI). However, it can be solved by enabling communication objects that encapsulate and standardize the communication functionalities for application-specific needs.

For satellite systems it is more convenient to adopt an object-oriented architecture since it behaves as a stateful system, meaning that satellites store data locally to keep control of current state. In fact, for the AOCS case studies the system is modelled using state vectors. That makes the object-oriented architecture more suitable for describing the software of a spacecraft.

This dissertation proposes adopting an agent-base software architecture model as an extension of the object-oriented architecture paradigm. The main intention is to provide capabilities to software components to handle their dynamic nature onboard a satellite.

1.5. AN OVERVIEW ON MULTI-AGENT SYSTEMS

Agent-based software engineering and multi-agent systems technologies are ideal to deal with the increase of complexity of satellite onboard software from an architectural point of view. This software architecture paradigm provides the required abstractions to integrate intelligent behaviors on satellite's software components. However, it is required to identify and provide the theoretical foundations as well as the tools to develop satellite onboard software applications using this approach.

It is necessary to establish the difference between intelligent agents and multi-agent systems. This section reviews the basic concepts related to multi-agent-based architectures, as well as their implementation considerations, so later they can be adopted to describe algorithms and methodologies proposed on this dissertation.

1.5.1. AGENTS VS MULTI-AGENT SYSTEMS

Initially, it is necessary to understand what agents are, and why they are so important to enable flexible software architectures. There is no clear consensus about the definition of agents. For this thesis purposes, a general definition provided by Russell et al. (1995) is considered. It establishes that an agent is: **"anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."**

This definition is broad and requires the refinement of some concepts to adapt it to the aerospace applications field. Also, it is necessary to establish the scope of these concepts to the engineering of onboard software for satellites. Some key elements have to be described in detail. Firstly, it is assumed that software systems can be decomposed into smaller routines that can be encapsulated into software agents. By definition, agents are social entities, so that they require interacting with each other to achieve their goals. Secondly, agents can interact with their environment. That makes the communication capabilities a key requirement for their success. Third, agents can have internal states that influence their decision-making process. Fourth, agents have the means to change their internal state.

Another aspect to consider is that agents could be either physical or virtual entities. Their location defines this status. Software Agents residing in the same logical container and sharing the same execution resources are considered virtual agents, whereas agents that interface with hardware elements are considered physical agents. The combination of physical and virtual agents defines a Multi-Agent Systems (MAS)

Similarly to agents, there are various definitions of multi-agent systems. Stone and Veloso (2000) describe a MAS as a **"loosely coupled network of problem-solving entities (agents) that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity (agent)."**

This definition of MAS implies that agents within the system have some communication to cooperate and achieve a common goal. As discussed by Glavic (2006), there are several typologies of cooperation for different types of agents. These included independent and cooperative agents that can achieve consensus using mechanism such as negotiation, deliberation, and emergent cooperation. From that work, one can conclude that studying the communication and organization aspects is fundamental to understanding the performance of Multi-Agent Systems and proposing design improvements.

Multi-agent systems have several architectural characteristics tailored to the applications domain where they are implemented. These include the ability to learn, negotiate, and cooperate to achieve common goals. According to agent modeling theory described in Elsenbroich and Gilbert (2014), in Multi-Agent Systems the starting point of formalization is a distributed, and concurrent system of agents; each of them solving a part of a larger task, which represents in our case functions of the onboard software of a space system.

Modeling software as a MAS requires a robust front-end systems engineering. Laouadi et al. (2014) describe a novel modeling methodology using formal specifications to requirements with Agent Unified Modeling Language (AUML). The core element of MAS-based software architecture is the organizations of agents. Jennings (2000) also describes the internal agents structure and its organization including specifications of their concurrency model, timing, and the relationships between software components.

As discussed by Jennings (2001), when adopting an agent-oriented view for software architecture design, it becomes apparent that most applications involve multiple agents to represent their decentralization and their concurrent nature. Shehory and Sturm (2014) establish that from a software architecture viewpoint, multi-agent systems modeling has two perspectives: one focused on internal agents structure, and the external perspective dealing with agents organizations and infrastructure services. The modeling, simulation, and deployment of MAS-based software architectures require methods and tools, which are commonly provided by platforms and frameworks.

1.5.2. AGENT COMMUNICATION ARCHITECTURES

Communication architectures and languages are key to facilitate the creation of interoperable multi-agent based software. One key requirement for agent's communication language is the decoupling of its behavior implementation from its interaction interface. For that purpose, two main topological approaches are considered: centralized and distributed. In the centralized approach, the Multi-Agent Systems platform offers communication and management capabilities encapsulated in dedicated platform agents. These are the Agent Management System (AMS), the Directory Facilitator (DF), and the Agent Communication Channel (ACC). For instance, the SPADE platform presented by Gregori et al. (2006a) implements a Extensible Messaging and Presence Protocol (XMPP) server for routing all the messages in the platform. The XMPP server acts as both the Agent Communication Channel and the Internal Platform Message Transport (IPMT) described in the communication reference model of the Foundations of Intelligent Physical Agents (FIPA). For message transport, SPADE relies on the Hypertext Transfer Protocol (HTTP) and Extensible Messaging and Presence Protocol (XMPP), but it also supports additional protocols for interoperability with other multi-agent platforms.

Java Agent Development Framework (JADE) is another multi-agent systems platform providing an application programming interface for Java-based Multi-Agent Systems. Details of this platform are presented and discussed by Bellifemine et al. (2007). Communication in JADE is by default provided and controlled from the main logical container by using the HTTP protocol. JADE also uses transport technologies such as Java-Remote Method Invocation (RMI), Internet Inter-Orb Protocol (IIOP), and Common Object Request Broker (CORBA) for enabling its run-time environment. Additionally, JADE

also supports distributed communication architecture by implementing a "Main Container Replication" feature that allows other logical containers within the multi-agent system to take control over the Agent Platform (AP) as described by Bellifemine et al. (2005).

Typically, centralized architectures use a star topology, where the AMS, ACC and DF are support agents living exclusively in the main container while in a distributed architecture implementation, for instance in a mesh topology, AMS, ACC and the DF agents are replicated in all the logical containers to ensure fault tolerant capabilities.

It is important to remark that not all the multi-agent systems platforms support both centralized and distributed implementation topologies for the support agents. That is the reason why having a good software design methodology is necessary to select or to design a proper platform to ensure the reliability requirements are satisfied. One of the open problems that is addressed by this research is the optimal organization of agents within the subsystems. It takes the total cost of communication under certain organizational constraints to study and propose algorithms to minimize that cost. It also explores strategies for establishing the optimal organization configuration by analyzing the grouping mechanisms, for instance teams, congregations, and coalitions.

1.5.3. MULTI-AGENTS SYSTEMS FRAMEWORKS

Although intelligent agents have been around since late of 90's, their actual implementation has not still been adopted fully in the software engineering community and has never been carried out to develop the entire onboard software of a small satellite mission. There are benefits in terms of reliability and performance that this architectural style can provide for engineering more capable and reliable space systems.

MAS frameworks and platforms are widely discussed and compared in literature by Sturm and Shehory (2014), Wang et al. (2014) and Rousset et al. (2014). This subsection elaborates on the most important findings from these sources. A guideline for the assessment of agent-based platforms is presented by Kravari and Bassiliades (2015). This comparison considers the following criterion: platform properties, usability, operating ability, pragmatics, and security management.

Most of the modeling and simulation platforms are based on Object-Oriented Language (OOL) like Java, Python, and C++. The advantage of using OOL is the ability to create a layered architecture style, which enables code reuse and modular implementation, enhancing the scalability of multi-agent systems solutions.

A Multi-agent systems implementation for distributed software is shown by Bellifemine et al. (2000) with the Java Agent Development Framework platform. In this application, the author focuses on implementing an agent class and describe specific agent's tasks by writing one or more behavior subclasses. The key advantage of using JAVA language for implementing multi-agent systems is the broad support for Java Virtual Machine (JVM), which enables a smooth deployment with COTS technologies. An example of a successful implementation can be found in the work presented by Bergenti et al. (2014) with Android-based devices.

JADE is one of the most popular frameworks for multi-agent software implementation. However, there are other platforms like ZEUS by Fonseca et al. (2001), JIAC by Lützenberger et al. (2013), SPADE by Gregori et al. (2006b), THOMAS by Argente et al.

(2011), and FIOT by do Nascimento and de Lucena (2017).

A key characteristic of multi-agent systems is their social ability. For making the agents social, it is required to provide a set of services, conventions, and knowledge to facilitate interaction and communications. Sycara et al. (2003) introduces and describes a summary of general MAS infrastructure services. However, in this case, it is necessary to assess which of the services are required in OBSW development.

Safety critical systems like satellites require determinism in the execution of their software applications as discussed by Burns and McDermid (1994). This characteristic has to be considered when selecting or designing a MAS development framework. In this dissertation, the JADE platform developed by Bellifemine et al. (2007) is used as a reference to compare all implementations intended for embedded computers in the satellite, since its a mature framework with stable implementation and support.

1.5.4. MAS-BASED APPLICATIONS IN CONTROL SYSTEMS

Multi-agent systems have become a hot topic for the control community over the last decade. Several applications have been reported in the literature. For example, in distributed control for wireless sensor networks by González-Potes et al. (2016), swarm robotics by Liang et al. (2016) and distributed power systems by Dou et al. (2017). One common aspect that these applications share is the use of multiple physical agents to reach consensus on specific target control.

On the other hand, there are applications of MAS-based control with virtual organizations, where agents share computing resources within the same system to achieve a goal. For example, in the work by Oviedo et al. (2010), they propose a general layered architecture style with four types of agents. They are the teleoperator agent, the coordinator agent, the operator agent and the device agent. This architecture considered a distributed and concurrent environment for execution of the application. The configuration was similar to a space systems system communicating over a linear data bus. The following subsection focuses on the application of MAS-based technology for space systems.

1.5.5. MAS-BASED SOFTWARE IN SPACE APPLICATIONS

According to Hassani and Lee (2015), the challenges of developing MAS-based software architectures for space missions are driven by four major characteristics of the spacecraft:

1. Very low human intervention during the operations phase.
2. High reliability during the mission lifetime.
3. Project development constraints: cost, schedule.
4. Spacecraft operations involves concurrent activities among a set of tightly coupled subsystems.

Space agencies like ESA and NASA have introduced MAS-based applications to improve the autonomy of spacecraft. Example of autonomous software for space applications was done in the Project for On-Board Autonomy (PROBA) mission and Livingstone

project. Detailed information for these applications can be found in the work of Hinchey and Vassev (2012) and Chien et al. (2014). From the literature review, application of multi-agent systems in space software has been demonstrated mainly in planning and scheduling algorithms for formation flying and control as in work of Chien et al. (2014). Also in applications for Telemetry, Tracking and Control (TT&C) by Wang et al. (2015) and Electric Power Subsystem (EPS) management by May and Loparo (2014). Based on the literature review, there is not any application of MAS-based software for the Attitude and Orbit Control Subsystem, which is the most time-critical subsystem within the spacecraft bus.

1.6. ENABLING TECHNOLOGIES FOR MAS-BASED SOFTWARE

In order to implement MAS-based software onboard satellites subsystems it is required to have a set of enabling technologies. This section discusses these technology enablers to make MAS-based software architectures feasible for their implementation.

1.6.1. MULTI-AGENT SYSTEMS INFRASTRUCTURE

Organizations with multiple software agents (MAS) require having an execution infrastructure that supports simultaneous task processing and communication. Concurrency and distribution can be achieved by dividing the processing time into slices and allocating them to different execution threads, or they can rely on multi-core processing for enabling space partitioning as well.

Gasser (2000) discusses the need to address Multi-agent Systems infrastructure as a technology enabler for agent-based software architecture adoption in more application fields (e.g Satellite systems). The elements needed for a robust multi-agent system ecosystem are grouped into 5 categories: system elements, services, capabilities, attributes, and community.

SYSTEM ELEMENTS

These are critical components that enable the implementation of agent-based applications. These include the way agents communicate to each other, the programing languages and libraries, design methodologies, experimental platforms with documented case studies, integrated development environments for developers, software developer kits, and finally implementation frameworks. Elements must be developed, tested and qualified for each application domain, in particular for those that are safety critical, for instance application in aerospace engineering.

SERVICES

Services are provided by underneath layers of the software stack, for instance the operating system. Services must provide capabilities for resource discovery and access control, security features, communication certification, as well as specialized features for domain specific needs.

CAPABILITIES

Capabilities refer to features that multi-agent systems infrastructure provides to systems engineers for analysis, trade-off, design, implementation, verification and validation.

Examples of capabilities include data collection tools (e.g for message exchange), fault detection, isolation and recovery, performance measurements, visualization tools, simulation environments, deployment analysis tools among several others depending on the application domain.

ATTRIBUTES

Attributes are characteristics of the MAS infrastructure to implement capabilities and services within the available resources. These includes aspects like openness, code density, robustness (fault-tolerance), scalability, standardization, availability among several others defined on application basis.

COMMUNITY SUPPORT

Community support is key for the long-term sustainability of the system. It is a special attribute that refers to the support environment for the MAS infrastructure. For example how the community shares its knowledge, how the user groups interact and which communication mechanisms they use, and also, how the components are distributed, for instance, using open source repositories (e.g GitHub).

1.6.2. MAS-BASED SOFTWARE DESIGN CONSIDERATIONS

So far, this Section has focused on general aspects that must be considered to adopt and adapt a Multi-agents system infrastructure for aerospace applications. There exist specific aspects that have to be discussed in detail due to safety-critical nature of satellite systems.

ALGORITHM FEASIBILITY

Multi-agent system formation is field of study devoted to research the possibilities of agents to team up to achieve a common goal. Depending on whether agents are physical or logical they are subject to satisfy specific requirements. For example, physical agents such as robots, are required to satisfy their kinematics and their logical constraints.

Tabuada et al. (2001) argue that the feasibility of an algorithm to be implemented as Multi-Agent Systems is defined by the formation graph that describes both individual agent kinematics and global inter-agent constrains. In that regard, two different types of formation graphs can be obtained: undirected and directed.

In undirected formations agents are equally responsible to maintain their constraints, whereas for directed formation particular agents are necessary to keep track and control of constraints. This paper also proposes algorithms to test the feasibility of a group of agents to achieve both kinds of formations.

DISTRIBUTED COMMUNICATION

Communication is one of the core elements that enables the implementation of multi-agent based software; However, its implementation is rather complex and requires application specific optimization. Li and Kokar (2013) establishes that the best way of dealing with this complexity is by introducing abstraction levels in its design. Agent communication protocols are required to specify a minimum set of rules for agent communication within the multi-agent system boundaries. There is also consensus on the importance of autonomy as part of agent communication semantics.

FIPA establishes the requirements for agent communication language as a set of standards that shall be met by multi-agent systems platforms. Fipa (2002) specifications describes the reference model for message transport, in which several communication requirements and constraints are documented, as well as a message structure and reference communication architectures. Communication protocols within multi-agent systems are grouped into layers intended to organize and isolate specific sets of functions. Gregori et al. (2006a) describe features like multi-user conferences are relevant when requiring advanced architecture configuration for self-organizing systems.

SOFTWARE DESIGN METHODOLOGY AND OPTIMIZATION

The development process of spacecraft is commonly divided into project phases. Within these phases, satellite software is designed, implemented, verified and validated using the traditional V-model approach.

According to Schaus et al. (2010) this process is centered on the use of documents, which means that at the end of each design phase the team has to focus on the review process, losing focus on the continued development process. This causes delays in the project execution timeline that are associated with increased cost. That situation requires a paradigm shift to novel and agile software development methodologies to enable flexibility in the design process, while coping with strict tight development schedules. The role of flexibility in system design for aerospace systems is discussed by Saleh et al. (2003), focusing mainly in the need of handling change in requirements during the development process.

Model-Driven Engineering (MDE) offers a framework to describe satellite systems using methods and techniques to optimize the requirements analysis and design of onboard software applications. According to Degueule et al. (2017), model-driven engineering provides with a unified and standardized language to implement models that are easy to read and explain to both decision makers and developers. With respect to complexity, MDE facilitates the handling of implementation details by establishing abstractions concepts and separation of concerns.

Combining model-driven engineering with multi-agents systems offers an opportunity of dealing with the increased satellite software complexity as described by Gascueña et al. (2012). However, this requires an extra effort of defining a methodology for end-to-end onboard software design, implementation, and verification. Several methodologies have been documented to develop agent-based software. For example, Prometheus by Padgham and Winikoff (2002)-Padgham et al. (2014), GAAIA by Zambonelli et al. (2003), Ingenias by Pavón and Gómez-Sanz (2003), MaSE by Deloach (2004), and several others offer MDE methodologies for Multi-agent Systems design, but they lack continuous verification and validation capabilities, necessary for enabling space software applications.

1.7. MOTIVATION AND CONTRIBUTIONS

So far, this Chapter has described the evolution of spacecraft computers, the trends in small satellite engineering, and the increase in satellite's onboard software complexity derived from the integration of FDIR features. Also, it proposed that one way to deal with the increment in the onboard software complexity is through the use of advanced architectural approaches such as those based in Multi-Agent Systems (MAS).

One question frequently asked is: why is it convenient to adopt an MAS-based approach for dealing with the software complexity problem? Also, there is a recurrent question about what is the difference between using a MAS-based approach and a multi-threading approach that is already implemented in most of the operating systems used for On-Board Software development. The first question is motivated by the connection of agent-based architectures with design patterns for implementation that enables faster development cycle. The second question is addressed by the implementation technique adopted in the making of the onboard software.

The focus of this dissertation is on the design of MAS-based software architectures for small satellites. The main aim is to improve mission autonomy by integrating the failure detection, isolation, and recovery of time-critical subsystems aboard satellites, for instance, the Attitude Determination and Control Subsystem (ADCS). It includes onboard software design through a MAS-based methodology that covers for agent modeling, simulation, and agents organizational optimization. For that purpose, it is necessary to establish the technologies and services required for their implementation. This section particularly focuses on describing the formulation aspects to cover during the research project. It includes the motivation, requirements, research focus (questions), the methodology to finally propose the structure followed during the research development.

1.7.1. RESEARCH MOTIVATION AND REQUIREMENTS

In Section 1.2 this research identified the trends and challenges of small satellite engineering. The integration of FDIR features was considered as a key feature for enabling autonomous operations of future small satellite missions. It was also demonstrated that the increment on the satellite's onboard functionalities is connected with a higher level in the onboard software complexity. That situation demands the adoption and development of innovative onboard software architectures. In order to address this problem, it was necessary to investigate prospective alternatives for the integration of FDIR capabilities at subsystem and/or component level. Additionally, it was required to define faster development and implementation workflows for these new approaches.

Since this research follows a systems engineering approach, some high-level requirements were defined to drive and constraint the research efforts. The main goal was to have a clear scope of the project while developing it. These high-level requirements were defined as follows.

The top-level requirements for this research are:

OBSW-ARCH-01: The software architecture shall maximize code reuse and code modularity (Killer requirement).

OBSW-ARCH-02: The software architecture shall optimize the use of onboard computing resources.

OBSW-ARCH-03: The software architecture shall enable the use of model-based development methods.

OBSW-ARCH-04: The software architecture shall support built-in fault detection, isolation and recovery features for time-critical systems in satellites.

OBSW-ARCH-05: The software architecture shall be implemented using COTS technology and open source development tools for small satellites.

Section 1.4 has introduced and described the potential architectural paradigms that were considered during the early stages of this research. It was clear that in order to satisfy the killer requirement above, it was necessary adopting a software architecture paradigm that maximized code reuse and component modularity. For that reason, the Object-Oriented Architecture (OOA) was selected for further research. However, one of the problems of using a purely OOA is that it does not have built-in FDIR features. This requires an extension of this architectural paradigm that enables the implementation of such features. That is the motivation to adopt a multi-based software architecture, since it allows extending OOA capabilities for satellite software development.

This research proposes the adoption of a MAS-based approach for on-board software architecture design with the purpose of enabling mechanisms to handle increased software complexity in time-critical subsystems of satellites. The Attitude Determination and Control Subsystem (ADCS) of small satellite missions is taken as a case study for verification and validation of the algorithms, models and methodologies proposed in this dissertation. The reason behind focusing on this subsystem is its time-critical nature, as well as its increasing requirements on reliability and performance in the coming future. All that is reflected in the ADCS complexity that is increasing to support newer satellite missions. These new applications demand new programming and processing paradigms to ease their implementation.

1.7.2. RESEARCH QUESTIONS

Based on the research motivation and requirements, the following research questions were defined for this dissertation:

1. What kind of services shall the onboard software provide for implementing highly reliable MAS-based software architectures in ADCS computers?
   (a) What is the best strategy for MAS-based FDIR implementation in small satellites?
   (b) What are the most critical services that onboard computers shall provide to implement MAS-based software architectures in ADCS computers?
   (c) How to balance services workload for an efficient MAS-based OBSW architecture implementation?
2. How to model the ADCS software architectures for small satellites using a multi-agent systems approach?
   (a) What kind of agents are required to describe ADCS software design?
   (b) What is the most effective methodology to describe agent's behavior in a multi-agent systems simulation environment?
   (c) What is the most suitable strategy to implement an ADCS software architecture using state of the art multi-agent systems platforms and tools?
3. How to optimize multi-agent system organization according to ADCS mission requirements and constraints?

(a) What kind of topology can be achieved by optimizing the performance of ADCS applications?

(b) How to link stakeholders requirements with quality attribute requirements in MAS-based ADCS software architecture?

(c) What are the risks, sensitivities, and trade-offs that must be considered and controlled to achieve an optimum MAS agent's organization in ADCS software architecture to satisfy mission requirements and constraints?

In the first research question, the focus is on the technical aspects related the multi-agent system implementation in the onboard computer of the satellite. Two key elements are being addressed: Firstly, the interface between the software application and the operating system of the onboard computer. For that purpose, a software library was developed to assess performance in memory management, CPU load and fault detection, isolation and recovery features. Then, the communication aspects of MAS-based application's implementations was assessed. From these activities, it was clear that communication is the key element required for the implementation of distributed and concurrent onboard software on satellite subsystems, thus an enabling feature.

The second research question is oriented to study the way that agents shall be engineered for satisfying performance and safety requirements in a satellite mission. Also, in this part, different tools are explored for modeling, simulation, and implementation of agent-oriented architectures for space applications. Modeling is one of the central elements of this thesis since it enables the development of design workflow that integrates the fault detection, isolation and recovery features for multi-agent based software.

The third research question explores and proposed novel algorithms for establishing organizations of agents within the onboard software implementation. It defines a set of cost functions to optimize the design architecture and study methods that can address constraints in reliability and performance of the software. Finally, this dissertation shows the potential of MAS-based software architectures to deal with constraint resources and uncertainty in the environment to provide more robust mission operations.

Before describing the processes and methods considered for the execution of this research, it is important to clarify the scope and focus taken to tackle the software complexity problem in satellite missions. Figure 1.4 summarizes the three main areas covered by this thesis that were introduced in previous sections. On one side, there is a set of multi-agent systems technologies that were analyzed and used as a baseline to extend their use to space systems. From the application perspective, the use of time-critical subsystems as case study supported the need for a new computing model for implementing complex pieces of software onboard the spacecraft.

From the literature reviewed, it was determined that Attitude and Orbit Control Subsystem (AOCS) was the most computing intensive subsystem on the spacecraft. Therefore, it is valuable for demonstrating MAS-based software architectures. Finally, the requirements specified built-in fault detection, isolation and recovery capabilities in the design of onboard software for satellite mission as key enabler for onboard autonomy in satellites. Based on this scope, this dissertation connects MAS, ADCS and FDIR during the on-board software development process with the purpose of improving performance and reliability of space missions.

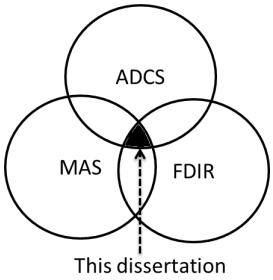


Figure 1.4: Dissertation Focus

1.7.3. RESEARCH METHODOLOGY

The general approach taken in this dissertation included both qualitative and quantitative aspects. It makes use of case studies with the ADCS subsystem to validate the research results. According to Runeson and Höst (2008), case studies are suitable for software development research since they allow studying the system in its natural context using both qualitative and quantitative tools. Initially, a qualitative study based on research papers surveys was performed for each of the research questions to identify and assess modeling techniques and tools that can be adapted for conducting experimental case studies with the ADCS of a satellite. These models were used for generating analytical and numerical models with controlled simulation experiments that were verified and validated against reference models from the literature using quantitative methods. (e.g., statistical tests, and linear regression)

The first research question is divided in two parts: One addressing the applicability of MAS-based architectures on FDIR algorithms for onboard software of ADCS subsystems. The second part focused on the services required for MAS-based software implementation. A review and comparison of FDIR methods was conducted to address the first part, as well as the design of an agent-based algorithm for detecting and correcting drift on gyroscopes used for ADCS implementation. The algorithm was verified numerically and validated against a linear regression model.

To address the second part on services required for the implementation of reliable onboard software, a qualitative analysis of existing MAS development frameworks was performed. From this comparison, four key components were identified, namely agents, messages, organizations, and platforms. Based on the interaction of these components two main capabilities for behavior allocation and communication were required to be studied in detail. The main effort was put on the implementation of the communication data bus that enable agents interactions in a linear bus topology often found in satellite implementations. For that purpose, an analytical model was derived to quantify the data bus load as a function of implementation and operation parameters. The analytical model was verified and validated against a discrete time simulation model controlling both implementation and operation parameters for an ADCS case study.

The same methodological approach was followed in research question 2. In the first part, a comparison of model-driven software development methodologies was performed to identify design gaps, but also to understand how these model-based methods

fit into the software development cycle of space systems. Once the gaps were identified, a development methodology was proposed to integrate end-to-end activities for model-driven software development with multi-agent systems for satellites. The quantitative experiments compared the implementation of attitude estimation algorithms using the proposed methodology and compared the numerical results to simulation models for verification and validation purposes.

In order to address the third research question, the software development workflow produced in the research question number two was refined to focus on the organizational aspects of MAS-based software. Again, a qualitative approach was used to describe organizational structures. Based on the qualitative analysis two main topological concepts were identified for its implementation in an ADCS case study. These topologies structures included risk analysis, and the definition of sensitivity and trade-off criteria to satisfy on-board software requirements.

An objective function was determined for optimizing the cost of communication within the organization of the MAS-based onboard software. The definition of that objective function required the establishment of constraints for tailoring it to the on-board software operations context.

For validation purposes, the optimization problem was explored using three different methods. The comparison of these methods allowed to assess their efficiency and performance, as well as their scalability. Also, it enabled the visualization of interaction between agents in the organization, which facilitates the implementation of MAS-based software.

1.8. DISSERTATION STRUCTURE

This dissertation is composed of seven chapters. Three are supporting chapters and four are core chapters devoted to present the research findings of this PhD research. Figure 1.5 depicts the relation of the chapters and the research questions defining a thesis work flow. Each of the core chapter addresses in detail aspects connected to a specific research question.

Chapter 1 describes the context of the research project, motivates the research problem and research questions. Also it gives an overview of the selected technology to address the software complexity problem in small satellite systems.

Chapter 2 provides the fundamentals of attitude and orbit determination that are used later in the core chapters as case studies for satellite applications. It also lays out the physical and mathematical formulation required as input for the analysis and design of improved software architectures with multi-agents systems.

Chapter 3 is devoted to research the feasibility of MAS-based algorithms to implement FDIR functionalities in the ADCS of a satellite. It makes a review of methods for FDIR in control systems and proposes an algorithm using an agent-based architecture that allows flexibility and modularity for its implementation. It presents numerical simulations to characterize its performance. It uses time-depend faults such as drift for gyroscopes used in ADCS implementation for small satellites.

Chapter 4 provides a description of the most critical services for implementation of multi-agent based software onboard of satellite systems. It is divided in two parts. The first part is used to describe the services required in OBC to implement MAS-based

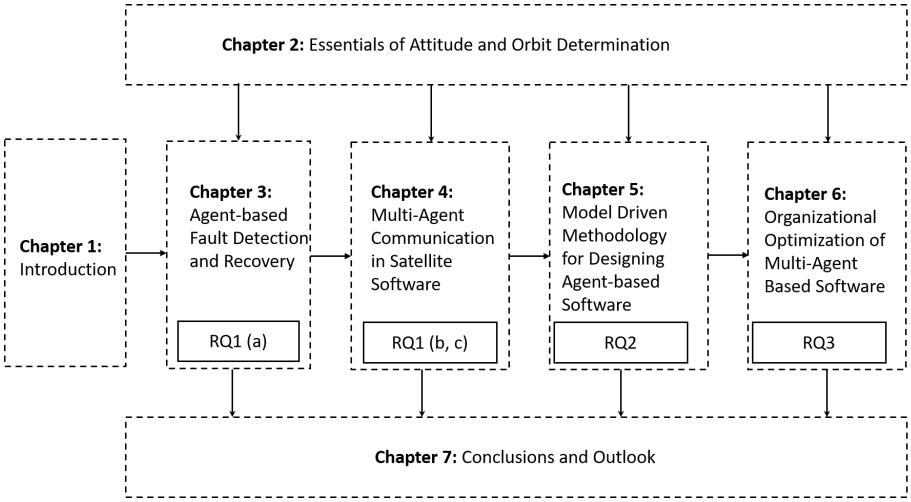


Figure 1.5: Ph.D. Thesis Work Flow

OBSW. The second part presents a characterization of the communication workload for extreme ADCS operations scenarios involving high information throughput on the data bus of the satellite.

Chapter 5 is focused in proposing a model-driven methodology for designing agent-based software in satellites. This method is called MASSA which stands for Multi-Agents Systems for Satellite Applications. This Chapter describes how to model on-board satellite software as a multi-agent system, and proposes a meta-modeling framework for using this approach in satellite missions. The methodology consists of four stages with feedback loops for optimization and verification.

Chapter 6 is focused on describing optimization aspects for multi-agent based software. Its main objective is to provide methods and tools for optimum topology design considering performance constraints. The research of this Chapter is intended to address organizational aspects of the agent-based software, to make them fit into the software development cycle of satellite missions, especially the implementation budgets.

Finally, Chapter 7 provides a critical reflection on the products of this research. It summarizes the findings that enhance the current body of knowledge. Besides the conclusions, this chapter offers a set recommendations for further research work and outlook.

2

ESSENTIALS OF ATTITUDE AND ORBIT DETERMINATION

"A single conversation with a wise man is better than ten years of study"

Chinese Proverb

Abstract

The purpose of this Chapter is to describe the concepts needed for the design of MAS-based software onboard of satellite subsystems. The Chapter focuses on the attitude and orbit control subsystem, which is used as the case study on this dissertation. The first Section describes and motivates a systems architecture for an agent-based AOCS using a cyber-physical system's approach. Then, it elaborates on the modeling of satellite orbit and attitude. That includes defining references frames, establishing an orbital model and describing the orbital dynamic of a spacecraft in the lower Earth orbit. Then, the basic models used to represent satellite attitude are introduced. These models are required to process the information coming from attitude sensors and providing a proper attitude state vector to the attitude controller. System's and measurements' disturbances are also introduced and described. Altogether the models provide the foundations in the establishment of a concurrent and distributed execution architecture for MAS-based state estimation used in later Chapters as a case study.

2.1. SYSTEMS ARCHITECTURE APPROACH

There is a trend in the space industry to design and build smaller satellites buses that maximize concurrency of onboard data processing of space missions. That increases satellite's complexity and requires the adoption of innovative approaches for enabling those computing capabilities.

Spacecraft can be described as a Cyber-Physical System (CPS) using the definitions proposed by Baheti and Gill (2011). In their work, they include the development of next-generation space vehicles as one of the key applications that can benefit from this approach. The main advantage of adopting a CPS-like architecture for space systems is their native capacity to support model-based development methodologies, that makes it desirable to work with highly constrained and complex systems.

According to Lee (2008) a CPS is defined as the integration of computation and physical processes in such a way that they feedback each other. The implementation of a CPS requires embedded computers and networks to monitor and control the physical processes. There are major challenges in CPS design and implementation, especially because the physical components introduce different reliability requirements and constraint from those needed in general-purpose computing. For this dissertation, a CPS-based system architecture is introduced in Figure 2.1 (see next page) to describe the AOCS of a satellite.

The proposed systems architecture groups the subsystem's elements into four categories related to their nature and position within the AOCS. These are the physical and logical domain and the back and the front end of these domains from the operations perspective. The onboard software components are colored purple for reference, and the onboard software application for AOCS is highlighted in bold types. Other systems components are colored different to make a distinction among them.

The physical domain includes the dynamic and kinematic models for the spacecraft, as well as the hardware components, for instance, sensors, actuators, onboard processors and harnessing to the spacecraft buses. The cyber domain refers to the software elements of the system.

These include simulation models, embedded firmware, sensor and actuator software drivers, operating systems and all the supporting libraries for MAS-based software execution. On top of that runs the AOCS software application developed to control its orbit and attitude according to particular mission's requirements. The front-end refers to connect of the AOCS with the spacecraft communication data bus, while the back-end elements interface the AOCS with the physical environment.

The next section provides the essential concepts of attitude and orbit control required for the implementation of onboard software of satellite systems. These models are used as input in the development of agent-based algorithms in Chapter 3, the study of communication effects in the AOCS estimation performance in Chapter 4 and the implementation of a model-based methodology for agent-oriented software design in Chapter 5, and the optimization of MAS organizations in Chapter 6.

2.2. AOCS MODELING CONCEPTS

This Section provides the essential theory required to describe the attitude and orbit of a satellite in the Lower Earth Orbit (LEO). It presents only the basic concepts required to understand the physical domain of the AOCS case study used in later Chapters of this dissertation to illustrate the applicability of MAS-based software architectures for fault-tolerant attitude estimation algorithms. Orbital elements concepts are described in Appendix A.

2.2.1. REFERENCE FRAMES

The motion of rigid bodies such as satellite requires the use of several reference frames to describe either their orbit and orientation, i.e. attitude. In this dissertation, five different reference frames are considered for that purpose as shown in Figure 2.2. These reference frames are based on the work of Jensen and Vinther (2010) for the AAUSAT3 and they can be re-used for describing the orbit and attitude of multiple Lower Earth Orbit satellites.

These are the Earth-Centered Inertial (i) frame, the Earth-Centered Earth-Fixed (e) frame, the Orbital (o) Frame, the Satellite-Fixed Body (s) frame and the controller (c) reference frame. They are briefly described here, but a more detailed explanation can be found in Montenbruck and Gill (2012) and Jensen and Vinther (2010).

The motion of rigid bodies is best described making use of an inertial reference frame. That is necessary for establishing an environment in which time and space behave homogeneously, isotropically, and in a time-independent manner. For that purpose, an

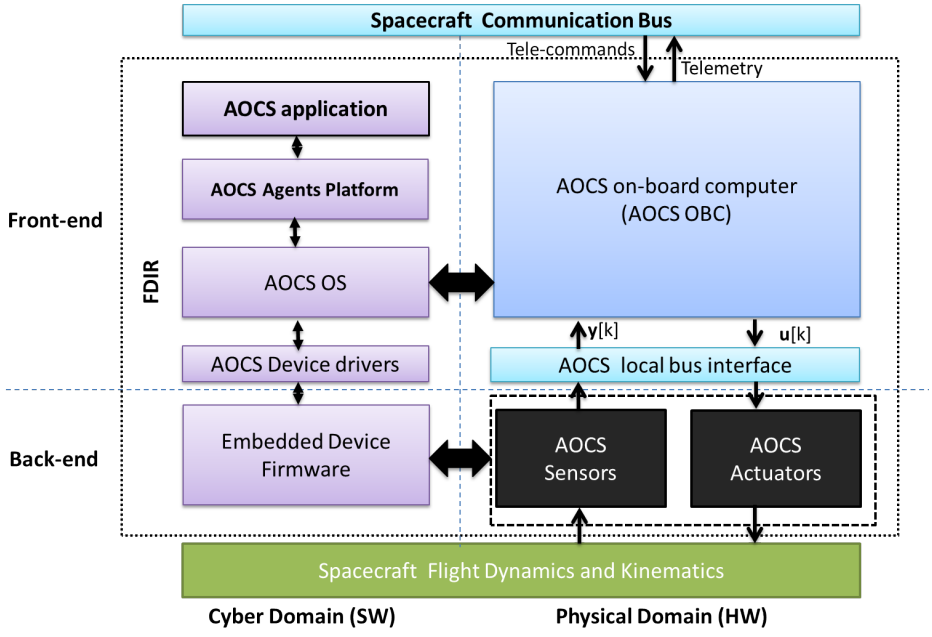


Figure 2.1: System architecture proposed for a MAS-based AOCS subsystem using a CPS approach.

Earth-Centered Inertial (ECI) is needed. It is placed in center of Earth with the x-axis ${}^{i}X$ pointing towards the intersection between the vernal equinox and the equatorial plane, and the z-axis ${}^{i}Z$ lies at a 90 degree angle to the equatorial plane. The y-axis ${}^{i}Y$ is the cross product between the x-axis and z-axis. The ECI is not a perfect inertial frame due to Earth's motion around the Sun, and other effects such as the Earth's nutation. However, these effects can be neglected for practical cases as discussed by Serway and Jewett (2018).

The Earth-Centered Earth-Fixed (ECEF) reference frame is needed for defining the magnetic field vectors and modeling ground tracking of satellites. The ECEF origin is in the Earth's center with the x-axis ${}^{e}X$ going through the intersection point of the Greenwich meridian with the equatorial plane. The z-axis ${}^{e}Z$ points toward the geographic north pole, and the y-axis ${}^{e}Y$ completes the right handed Cartesian coordinate system.

The Orbital Reference Frame (ORF) defines its z-axis ${}^{o}Z$ always nadir pointing, while its x-axis ${}^{o}X$ follows the direction of the satellite's velocity vector. The y-axis ${}^{o}Y$ is defined by the cross product of the z-axis and the x-axis. It is originated at the satellite's Center of Mass (CoM).

The Controller Reference Frame (CRF) is also originated at the CoM of the satellite with its x-axis ${}^{c}X$ pointing along the minor axis of inertia and z-axis ${}^{c}Z$ pointing through the major axis of inertia. The y-axis ${}^{c}Y$, known as the intermediate axis of inertia, is calculated by a cross product of x-axis and the z-axis. The CRF is a body-fixed reference frame.

The Satellite Body-Fixed Reference Frame (SBRF) is mainly used to define the orientation of ADCS sensors and the measurements required to determine satellite's attitude. The axis are defined parallel to the satellite frame structure as shown in Figure 2.2 (d).

2.2.2. SATELLITE ORBIT MODEL IN LEO

The motion of a satellite in LEO can be modeled as

$$\ddot{\mathbf{r}} = -\mu_E \frac{\mathbf{r}}{r^3} + \mathbf{a}_d(t, \dot{\mathbf{r}}, \mathbf{r}, \mathbf{q}) \tag{2.1}$$

where, $\mathbf{r}$ is satellite's geocentric position vector in the ECI frame, μ_E is the gravitational coefficient of the Earth and $\mathbf{a}_d$ are perturbing accelerations of the satellite as a function of time t, velocity $\dot{\mathbf{r}}$, position $\mathbf{r}$ and the spacecraft attitude quaternion $\mathbf{q}$.

The perturbing accelerations $\mathbf{a}_d$ are primarily due to the higher order terms of the Earth's gravitational field, atmospheric drag, solar radiation pressure, solar and lunar tidal effects and general relativistic effects. More detailed models for precise orbit determination on satellites can be found in the work of Švehla and Rothacher (2003).

The satellite's initial conditions for position and velocity in the ECI at the reference epoch t_e can be described using orbital elements: semi-major axis a, eccentricity e, inclination i_o, right ascension of the ascending node Ω_a, argument of perigee ω_p and the true anomaly ν_a at epoch t_0 as described in Appendix A. More details on celestial mechanics are provided in Brouwer and Clemence (2013).

2.2.3. ATTITUDE REPRESENTATION

The attitude of a spacecraft is its rotational orientation in space with respect to a reference coordinate system (e.g. ECI), as described in Großekatthöfer and Yoon (2012).

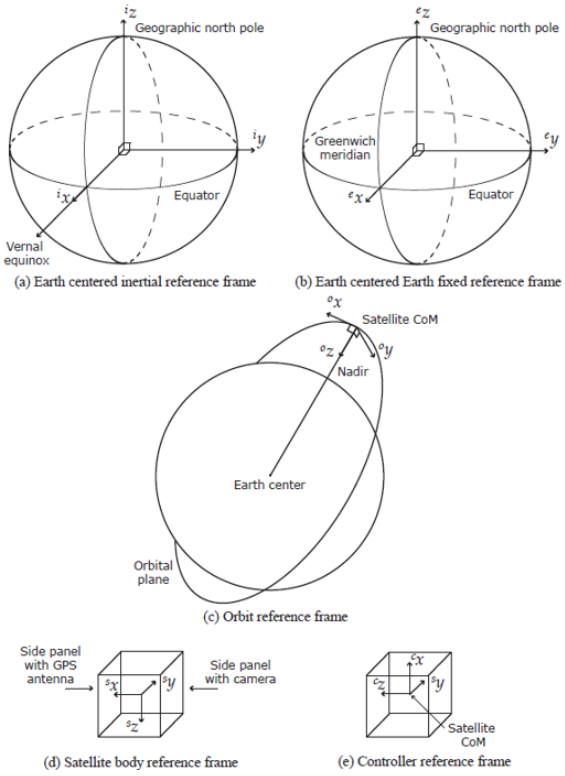


Figure 2.2: References frames used to model the orbit and attitude of AAUSAT: (a) Earth-Centered Inertial reference frame, (b) Earth-Centered Earth-Fixed reference frame, (c) Orbital Reference Frame, (d) Satellite Body-Fixed Reference Frame and (e) Controller Reference Frame. Source: Jensen and Vinther (2010)

There are various methods for the mathematical representation of a rigid body's attitude. The most commonly used are the Direction Cosine Matrix (DCM), Euler Angles and the Quaternion which offers advantages such as no singularity, and computationally less intense compared to DCM and Euler Angles.

Euler's rotation theorem states that for a three-dimensional space ***any displacement of a rigid body such that a point on the rigid body remains fixed, is equivalent to a single rotation about some axis that runs through the fixed point***. This theorem is used to describe the satellite kinematic models.

Due to above-mentioned advantages, the quaternion notation is used in this dissertation for representing satellite's attitude. A quaternion consist of four elements that are grouped in two parts. One that represents the attitude parameters $\mathbf{e}= [q_1, q_2, q_3]^T$, which is called the vector part, and the last element q_4 that represents the scalar part s. This is

represented as

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \mathbf{e} \\ s \end{bmatrix}. \tag{2.2}$$

The quaternion notation is constrained by the relationship:

$$\mathbf{q}^T\mathbf{q} + q_4^2 = 1. \tag{2.3}$$

The coordinate transformation from system A to a System B can be represented using quaternion notation as

$${}^{B}_{A}\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \|\mathbf{e}\| \cdot sin(\theta/2) \\ cos(\theta/2) \end{bmatrix} \tag{2.4}$$

where, $\|\mathbf{e}\|$ is the normalized rotational axis and θ represents the transformation angle for these coordinate systems.

2.2.4. ATTITUDE MODELING

The attitude of a satellite is constantly changing over time so that the parameters to describe it are time-dependent. In Field and Pence (1984), the attitude models can be described by its kinematic and dynamic equations. In kinematics, the study of satellite motion is irrespective of the forces that cause it, while in dynamics the understanding of physical forces is needed to describe its behavior

KINEMATIC EQUATION

Suppose a satellite rotating over its Satellite Body-Fixed Reference Frame (s) with respect to the Earth-Centered Inertial (i) reference frame with an angular velocity $\boldsymbol{\omega} = [\omega_1\ \omega_2\ \omega_3]^T$. The orientation of the satellite can be obtained performing a frame rotation parameterized by the transformation matrix $\mathbf{A}(\mathbf{q})$. Using this transformation matrix, a measurement vector in the inertial frame ${}^{i}\mathbf{v}$ can be expressed in the satellite-fixed body frame as ${}^{s}\mathbf{v} = \mathbf{A}(\mathbf{q}){}^{i}\mathbf{v}$.

Then, the angular motion can be described using a differential equation as

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \tag{2.5}$$

where, $\boldsymbol{\Omega}(\omega)$ is defined as a $4x4$ skew symmetric matrix $\boldsymbol{\Omega}(\cdot)$ of the angular velocity as

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \tag{2.6}$$

and, the DCM is used to establish the transformation from Earth-Centered Inertial reference frame to the Satellite Body-Fixed Reference Frame in terms of the orientation quaternion as

$$\mathbf{A}(\mathbf{q}) = \frac{1}{\sqrt{\|\mathbf{e}\|^2 + q_4^2}} \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_4q_1) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_4q_1) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}. \quad (2.7)$$

DYNAMIC EQUATION

According to Wie (2008), spacecraft can be assumed as rigid body with six degrees of freedom. Three of them are related to rotational motion and the other three to transnational motion (orbit). In an inertial frame, Euler's second law states that there is a relation between the change in time of the angular momentum $\mathbf{L}$ and the external torques $\mathbf{N}_{ext}$ applied to the spacecraft. That relation is presented in Wie (2008) as

$$^{i}\dot{\mathbf{L}} = {}^{i}\mathbf{N}_{ext}. \quad (2.8)$$

In the Controller Reference Frame the relation can be re-written as

$$^{c}\dot{\mathbf{L}} = -{}^{c}\boldsymbol{\omega} \times {}^{c}\mathbf{L} + {}^{c}\mathbf{N}_{ext}. \quad (2.9)$$

The angular momentum can be represented also using the inertia of the satellite $\mathbf{I}_{sat}$ along the CRF as

$$^{c}\mathbf{L} = \mathbf{I}_{sat}{}^{c}\boldsymbol{\omega}. \quad (2.10)$$

Substituting (2.10) in (2.9) generates

$$\mathbf{I}_{sat}{}^{c}\dot{\boldsymbol{\omega}} = -{}^{c}\boldsymbol{\omega} \times (\mathbf{I}_{sat}{}^{c}\boldsymbol{\omega}) + {}^{c}\mathbf{N}_{ext}. \quad (2.11)$$

The composition of the externally applied torques includes the addition of both control and disturbances torques, so that $\mathbf{N}_{ext}$=$\mathbf{N}_{ctrl}$+$\mathbf{N}_{dis}$. The dynamic equation of the satellite can be reorganized as

$$\mathbf{I}_{sat}{}^{c}\dot{\boldsymbol{\omega}} + {}^{c}\boldsymbol{\omega} \times (\mathbf{I}_{sat}{}^{c}\boldsymbol{\omega}) = {}^{c}\mathbf{N}_{ctrl} + {}^{c}\mathbf{N}_{dis}. \quad (2.12)$$

Finally, it can be rewritten to describe the change in the angular velocity as a function of the applied torques as

$$\dot{\boldsymbol{\omega}} = \mathbf{I}_{sat}^{-1}\left[-\mathbf{S}(\boldsymbol{\omega})(\mathbf{I}_{sat}\boldsymbol{\omega}) + \mathbf{N}_{ctrl} + \mathbf{N}_{dis}\right] \quad (2.13)$$

where, $\mathbf{S}(\cdot)$ for $\boldsymbol{\omega}$ is a $3x3$ symmetric skew matrix described as

$$\mathbf{S}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \quad (2.14)$$

ATTITUDE STATE MODEL

Previous subsections introduced the kinematic and dynamic models used to describe the attitude of a satellite. The kinematic equation relates satellite's attitude representation $\mathbf{q}$ and its angular velocity $\boldsymbol{\omega}$, while the dynamic equation describes the relation of the angular velocity and the external torques applied to the spacecraft. These equations can be combined to describe the state of the satellite as

$$\begin{aligned} \mathbf{x} &= \left[{}^{c}_{i}\mathbf{q}^{T} \;\; {}^{c}\boldsymbol{\omega}^{T} \right]^{T} \\ \mathbf{x} &= \left[q_1 \; q_2 \; q_3 \; q_4 \; \omega_1 \; \omega_2 \; \omega_3 \right]^{T}_{.} \end{aligned} \tag{2.15}$$

Combining (2.5) and (2.13) the change in time of satellite's state $\dot{\mathbf{x}}$ can be described as

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \\ \mathbf{I}_{sat}^{-1}\left[-\mathbf{S}(\boldsymbol{\omega})(\mathbf{I}_{sat}\boldsymbol{\omega}) + \mathbf{N}_{ctrl} + \mathbf{N}_{dis} \right] \end{bmatrix}. \tag{2.16}$$

The attitude measurement model $\mathbf{y}$ is defined as follows

$$\begin{bmatrix} \mathbf{q} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{I}_{7x7} \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\omega} \end{bmatrix}. \tag{2.17}$$

In Chapter 5 the state vector $\mathbf{x}$ is extended to accommodate fault mechanisms related to measurement bias for the magnetometer $\mathbf{b}_{mag}$ and the gyroscope $\mathbf{b}_{gyro}$ as

$$\mathbf{x}_{ext} = \left[{}^{c}_{i}\mathbf{q}^{T} \;\; {}^{c}\boldsymbol{\omega}^{T} \;\; {}^{c}\mathbf{b}_{mag}^{T} \;\; {}^{c}\mathbf{b}_{gyro}^{T} \right]^{T}. \tag{2.18}$$

2.2.5. ATTITUDE PERTURBATION MODELING

The environment in which the satellite operates has multiple sources of perturbations. These perturbations affect the quality of the systems models and lead to performance and reliability issues. This Subsection describes those effects related to environmental phenomena. These are the gravitational torque, aerodynamic torque, environmental radiation torques and the magnetic torque.

GRAVITATIONAL TORQUE

Satellites follows the Kepler's laws of planetary motions. However, due to Earth's irregularities on its mass distribution, tidal movements of the oceans and third body gravitational fields the satellite orbit is affected. This has to be considered as an input during the orbital design process. Robertson (1958) describes in detail how to derive its components along the principal axis of inertia of a spacecraft using the potential function for a small rigid body. This torque can be used to passively stabilize satellites attitude by controlling its initial orbital injection or deploying mechanisms that allows the generation of stable equilibrium points as described by Peter Hughes (2012).

AERODYNAMIC TORQUE

The boundary between the atmosphere and the space is defined in different ways. They could be artificially imposed, for instance by space law, or it can determined by physical parameters like the atmospheric drag force. Zagórski (2012) describes the effect of altitude, atmospheric density, drag coefficient, reference area and its velocity on the force acting upon a spacecraft. Apart from decaying orbit of the satellite, it is also observed that atmospheric drag introduces torques that modifies its attitude. This torque is found to depend on the distance between the center of mass and the center of pressure where the drag force is applied. Atmospheric profiles are often used to model the effect aerodynamic torque on spacecraft, for example the MSIS thermosphere model. Hedin (1991) proposes to extend these models to account more precisely for effects in the middle and lower atmosphere where several satellite constellations are intended to operate .

SOLAR RADIATION PRESSURE TORQUES

Environmental radiation in LEO orbits comes mainly from the Sun. Due to Earth's eccentricity it can vary seasonally. The main source of radiation torque is the direct Sun radiation, then due to indirect sunlight. Albedo and infrared radiation are known to also produce radiation torques as described by Shrivastava and Modi (1983). Spacecraft with large antenna elements, or with deployable mechanisms shall account for radiation torque effects, since they can bend or damage in the long term as shown by Etkin and Hughes (1967) for Aloutte I and Explorer XX satellites.

MAGNETIC TORQUE

The magnetic torque affecting a satellite depends both on its magnetic momentum and the external magnetic field surrounding it. For a spacecraft orbiting the Earth, the magnetic field can be approximated to a dipole field. Models like IGRF presented in the work of Macmillan and Maus (2005) provides periodically updated spherical coefficients to calculate values of magnetic flux density $\mathbf{B}$ for a specific point in time and space in a LEO. That makes the magnetic torque highly controllable in AOCS implementation. Hysteretic materials are often utilized in form of custom shaped rods to provide passive momentum damping and other attitude control mechanisms as described by Kim (2011).

2.2.6. SENSOR MEASUREMENT MODEL

Sensors are key components within the AOCS. They allows to measure the state of the spacecraft over time. However, their performance is affected by intrinsic and extrinsic factors. According Curey et al. (2004), the main sources of error in measurements are due to installation errors (scale factor), measurement drift, time delay, measurement bias, noise, and random walk. Also the measurement's range and the sensor resolution play a role in the quality of the obtained values.

The installation error, often called scale factor error $\mathbf{S}$, is quantified by establishing a static multiplicative parameter to calibrate the true signal value $\mathbf{v}(t)$ to the measured signal value $\mathbf{y}(t)$. It is used to compensate for potential problems in the sensor installation, mainly due to alignment issues.

The measurement drift $\mathbf{d}(t)$ is an additive error that can be modeled as a systematic linear effect (ramp) produced due to the integration of any random error (e.g walk,

noise). It is defined as the change in a measured value when it is measured under the same conditions after a period of time. Since it is a systematic effect it can be compensated using model-based methods, for example, Artursson et al. (2000) demonstrate a method for drift compensation implemented in gas sensors for chemical industry.

Time delay is very common in distributed communication systems used widely in recent space missions. It is modeled as a shift in time of the sensor response with respect to the real sensor measurement response $\mathbf{v}(t-\tau)$.

Bias $\mathbf{b}$ is a constant deviation of the measured value regarding its true value. It is also modeled as an additive error. It can be estimated when the output signal of a sensor (true value) is zero and compensated in the engineering model.

Noise $\mathbf{n}(t)$ is a random deviation of the true signal value varying over time. For Gaussian processes it is described mainly using its variance. The noise in the sensor models is represented by a random variable $\mathbf{r}(t)$ with zero mean and standard deviation σ.

Random walk $\mathbf{r}(t)$ in statistics describes a situation where the output of a system is driven by a succession of random steps. The random walk is modeled as the integral of a random variable with zero mean. In Stockwell (2003), random walk for rate sensors is determined as the average deviation that occurs when integrating the output signal to get the angle changes.

The effect of measurement resolution and range are not considered in this model since they depend on implementation aspects that varies from mission to mission. Also there are errors due to quantization and round off that can affect the precision of multisensor fusion. These are not consider in the measurement model used for this research.

Finally, (2.19) summarize the model to describe measurements in the simulation case studies for later dissertation chapters.

$$\mathbf{y}(t) = \mathbf{S}(\mathbf{v}(t-\tau) + \mathbf{d}(t-\tau) + \mathbf{n}(t-\tau) + \mathbf{r}(t-\tau) + \mathbf{b}) \tag{2.19}$$

2.3. ONBOARD ATTITUDE DETERMINATION

The AOCS is in charge to provide satellite operators with capabilities to change both its orbit and attitude. Although Chaves-Jiménez et al. (2017) describe orbit and attitude as coupled behaviors in LEO orbits. This dissertation takes them as independent processes (for simplification purposes), and it focuses particularly on attitude determination. For that reason this section describes the main concepts used in describing case studies to demonstrate the application of agent-based architectures in small satellite missions.

Onboard attitude determination methods mostly rely on complementary information provided by multiple sensors onboard the spacecraft as discussed by Wertz (2012). The fusion of multiple sensor data improves state observability. For orbit the most popular sensors are GNSS-based for instance GPS, but for attitude there is wide variety of sensors that can be used based on mission needs. For example, in LEO, the most common sensors are magnetometers, star trackers (e.g sun sensor), gyroscopes. The information coming from these sensors has to be combined (fused) to estimate spacecraft's position and orientation.

2.3.1. CHALLENGES ON MULTI-SENSOR DATA FUSION

Multi-sensor data fusion relies in the performance of physical sensors that not always behave as expected during the system design phase. Khaleghi et al. (2013) define four categories to describe issues with data fusion in multi-sensor systems. These are data imperfection, correlation, inconsistency and disparateness.

IMPERFECTION

Data imperfection refers to both internal and external causes that provoke loss of information in the measurements. Among these effects uncertainty play a relevant role. Sensors are subject to noise, distortion and interference that causes measurement's corruption. Also sensor's imprecision (e.g ambiguity and incompleteness) and granularity (e.g measurement data type) can affect the quality of sensor measurements. These limitations come mainly from technology aspects related to sensor's implementation.

CORRELATION

Correlation effects are product a mutual relationship or association between measurements. This effect is specially problematic in distributed fusion systems. It happens when the same information takes multiple paths from the source sensor to the fusion node or due information re-circulation from output of a fusion node back to the input (feedback).

INCONSISTENCY

It refers to non-systematic aspects that affect the integrity of the data fusion performance.That encompasses spurious, as well as disordered and conflicting data. They can be spacial, for instance outliers, or temporal like out of order arrival. Their main effect is in the estimation performance rather than the quality of the instant measurements. They can be associated to communication delays in distributed systems and communication channel saturation, which is addressed in Chapter 4.

DISPARATENESS

This category refers to the homogeneity between the information produced by the sensors within the systems. For example some sensors can produce images, while others produce audio, just like the human senses. The challenge here, is to find a common representation that can be handled by the processing units to implement the estimation technique accordingly.

Data fusion algorithm are expected to systematically address aforementioned issues by integrating features for sensors fault's detection and correction. The next two sections describe the most relevant methods used in space applications.

2.3.2. DATA FUSION TECHNIQUES

Several methodologies have been proposed in the literature for multi-sensor data fusion and information aggregation on intelligent systems. El Faouzi et al. (2011) classifies these methods according its data processing technique in three main categories: statistical, probabilistic and artificial intelligence based methods.

STATISTICAL METHODS

These methods include algorithms as simple as arithmetic means and regression analysis (e.g Least Squares used in the 60's for satellite state estimation by Wahba (1965)) to more sophisticated multivariate statistical analysis, and others used recently for data mining engines. This kind of methods are not very well suited for online state estimation, but they perform better in offline analysis.

PROBABILISTIC METHODS

Probabilistic data fusion take information from a sequence of past actions and combine the with local observations to integrate them into an estimation of the current state. Examples of probabilistic approach are Bayesian networks in Heckerman et al. (1995), maximum likelihood estimators in Rauch et al. (1965) and the Kalman filter family used for optimal state estimation by Simon (2006). Probabilistic methods are very well suited to deal with measurements imperfection and correlation in multi-sensor data fusion.

MACHINE LEARNING METHODS

Samuel (1959) defined Machine learning as a field of artificial intelligence devoted to provide computers-based systems the ability to learn without being explicitly programmed. That gives systems the capability of adapting to dynamic environments. Machine learning methods for data fusion in space applications includes reinforced learning presented in Van Buijtenen et al. (1998), and recently the adoption of deep belief networks by Li et al. (2017b). Machine learning combines statistical and probabilistic methods with available data to calibrate estimators and achieve a better performance. They also take advantage of available computing technology to speed-up the learning process. The main drawback preventing the adoption in online estimation is the complexity of model training and tuning that is currently yet under development.

2.3.3. REFERENCE ALGORITHM FOR ATTITUDE ESTIMATION

According to Crassidis et al. (2007), attitude estimation is a two-step process in which vehicle's orientation requires fusing body-frame measurements, reference observations and filtering of noisy sensor measurements. The filtering process can be achieved combining measurements with kinematic and dynamic models to complement each other.

Simon (2006) establishes that in the process of estimation with the Kalman filters, the Extended Kalman Filter (EKF) is the most preferred method due a good balance between performance and implementation complexity. This section will focus on describing the implementation of the EKF used as a reference algorithm for comparison purposes.

The implementation of the EKF follows the process described in Appendix C with the satellite equations introduced in Subsection 2.2.4. The EKF implementation is based on the work of Jensen and Vinther (2010) for the AAUSAT 3 spacecraft, since the sensors and the ADCS physical architecture matches the one presented as a case study in Chapters 3, 4, and 5. The EKF implementation takes the quaternion $\mathbf{q}$ representing the rotation of the spacecraft from ECI frame (i) to the Controller Reference Frame (c), as well as its angular velocity $\boldsymbol{\omega}$ in the CRF of the satellite. For that purpose, a state vector $\mathbf{x}$ is defined as

$$\mathbf{x} = \left[{}^{c}_{i}\mathbf{q}^{T} \;\; {}^{c}\boldsymbol{\omega}^{T} \right]^{T} = [q_1 \;\; q_2 \;\; q_3 \;\; q_4 \;\; \omega_1 \;\; \omega_2 \;\; \omega_3]^{T}. \tag{2.20}$$

The error state is defined as

$$\delta\mathbf{x} = \left[\delta\mathbf{q}_{1:3}^{T} \;\; \delta\boldsymbol{\omega}^{T}\right]^{T} = [\delta q_1 \;\; \delta q_2 \;\; \delta q_3 \;\; \delta\omega_1 \;\; \delta\omega_2 \;\; \delta\omega_3]^{T}. \tag{2.21}$$

Using the results from Bak (1999), the continuous time Jacobian matrix for the error state can be expressed as

$$\mathbf{F}(t) = \begin{bmatrix} -\mathbf{S}({}^{c}\bar{\boldsymbol{\omega}}) & \frac{1}{2}\mathbf{1}_{3x3} \\ \mathbf{0}_{3x3} & {}^{c}\mathbf{I}_{sat}^{-1}[\mathbf{S}({}^{c}\mathbf{I}_{sat}{}^{c}\bar{\boldsymbol{\omega}}) - \mathbf{S}({}^{c}\bar{\boldsymbol{\omega}}{}^{c}\mathbf{I}_{sat})] \end{bmatrix} \tag{2.22}$$

where, ${}^{c}\bar{\boldsymbol{\omega}}(t)$ is the nominal angular velocity in the CRF of the satellite, $\mathbf{1}_{3x3}$ is a 3x3 identity matrix and ${}^{c}\mathbf{I}_{sat}$ is the inertia matrix of the satellite given in its body frame.

According to the work of Bak (1999) it is also possible to express the discrete time Jacobian measurement model for the error state as

$$\mathbf{H}_k = \begin{bmatrix} 2\mathbf{S}({}^{c}\mathbf{v}_{sun,k|k-1}) & \mathbf{0}_{3x3} \\ 2\mathbf{S}({}^{c}\mathbf{v}_{mag,k|k-1}) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{1}_{3x3} \end{bmatrix} \tag{2.23}$$

where, ${}^{c}\mathbf{v}_{sun,k|k-1}$ and ${}^{c}\mathbf{v}_{mag,k|k-1}$ are the the predicted sun vector measurements $\mathbf{v}_{sun}$, and the predicted magnetic field vector measurement $\mathbf{v}_{mag}$ in the CRF, respectively.

The implementation of the EKF requires a set of inputs variables and constant parameters described in Table 2.1. Then pseudo-code for the implementation of the EKF is shown in the Algorithm 1.

Table 2.1: Input variables and parameters for the implementation of the EKF Algorithm

Name	Description	Type of Input
${}^{s}\mathbf{v}_{sun,k}$	Measured sun vector in the Satellite Body-Fixed Reference Frame	Variable
${}^{s}\mathbf{v}_{mag,k}$	Measured magnetic field vector in the Satellite Body-Fixed Reference Frame	Variable
${}^{s}\boldsymbol{\omega}_k$	Measured angular velocity of the satellite in its body frame at $t = k$	Variable
${}^{i}\mathbf{v}_{sun,k|k-1}$	Predicted sun vector in the Earth-Centered Inertial reference frame	Variable
${}^{i}\mathbf{v}_{mag,k|k-1}$	Predicted magnetic field vector in the Earth-Centered Inertial reference frame	Variable
E_f	Eclipse Flag	Variable
${}^{s}\mathbf{N}_{Cont,k}$	Applied control torque including torque from permanent magnet in in the Satellite Body-Fixed Reference Frame	Variable
${}^{s}\boldsymbol{\omega}_0$	The initial angular velocity of the satellite in its body reference frame	Constant
${}^{s}_{i}\mathbf{q}_0$	Initial quaternion representing the rotation of the satellite's body frame relative to the Earth-Centered Inertial reference frame.	Constant
T_s	Sample period between filter iterations.	Constant
$\mathbf{P}_0$	Initial error covariance matrix	Constant
$\mathbf{Q}$	Process noise matrix	Constant
$\mathbf{R}$	Measurement noise matrix	Constant
$\mathbf{I}_{sat}$	Satellite inertia matrix	Constant

Algorithm 1 EKF for Attitude Estimation (Jensen and Vinther (2010)

)

procedure ESTIMATION
Initialize:
  $\mathbf{x}_{k|k} \leftarrow \left[({}^{s}_{i}\mathbf{q}_0 \otimes {}^{c}_{s}\mathbf{q})^T \ (\mathbf{A}({}^{c}_{s}\mathbf{q}){}^{s}\boldsymbol{\omega}_0)^T \right]^T$
  Measurements $\leftarrow {}^{s}\mathbf{v}_{sun,k}, {}^{s}\mathbf{v}_{mag,k}, {}^{s}\boldsymbol{\omega}_k, {}^{s}\mathbf{N}_{ctrl,k}$
  Load $\mathbf{P}_0, \mathbf{Q}, \mathbf{R}, \mathbf{I}_{sat}$
Predict:
  Rotate Control Torque: ${}^{c}\mathbf{N}_{ctrl,k-1} \leftarrow \mathbf{A}({}^{c}_{s}\mathbf{q}){}^{s}\mathbf{N}_{Cont,k-1}$
  Priori State Propagation: $\mathbf{x}_{k|k-1} \leftarrow RK4(\mathbf{x}_{k-1|k-1}, {}^{c}\mathbf{N}_{ctrl,k-1}, T_s, steps)$
  Calculate Discrete Jacobian: $\boldsymbol{\phi}_{k-1|k-1} \leftarrow \mathbf{1}_{6x6} + T_s\mathbf{F}(\mathbf{x}_{k-1|k-1})$
  Calculate Priori Error Covariance: $\mathbf{P}_{k|k-1} \leftarrow \boldsymbol{\phi}_{k-1|k-1}\mathbf{P}_{k-1|k-1}\boldsymbol{\phi}^T_{k-1|k-1} + \mathbf{Q}$
Update:
  Read: ${}^{s}\mathbf{v}_{sun,k}, {}^{s}\mathbf{v}_{mag,k}, {}^{s}\boldsymbol{\omega}_k, {}^{s}\mathbf{N}_{ctrl,k}$
  if $E_f = 0$ **then**
    Normalize and Rotate SS k: ${}^{c}\mathbf{v}_{sun,k} \leftarrow \mathbf{A}({}^{c}_{s}\mathbf{q}) \frac{{}^{s}\mathbf{v}_{sun,k}}{\left\| {}^{s}\mathbf{v}_{sun,k} \right\|}$
    Normalize and Rotate SS k|k-1: ${}^{c}\mathbf{v}_{sun,k|k-1} \leftarrow \mathbf{A}({}^{i}_{c}\mathbf{q}_{k|k-1}) \frac{{}^{s}\mathbf{v}_{sun,k|k-1}}{\left\| {}^{s}\mathbf{v}_{sun,k|k-1} \right\|}$
    Normalize and Rotate Mag k: ${}^{c}\mathbf{v}_{mag,k} \leftarrow \mathbf{A}({}^{c}_{s}\mathbf{q}) \frac{{}^{s}\mathbf{v}_{mag,k}}{\left\| {}^{s}\mathbf{v}_{mag,k} \right\|}$
    Normalize and Rotate Mag k|k-1: ${}^{c}\mathbf{v}_{mag,k|k-1} \leftarrow \mathbf{A}({}^{i}_{c}\mathbf{q}_{k|k-1}) \frac{{}^{s}\mathbf{v}_{mag,k|k-1}}{\left\| {}^{s}\mathbf{v}_{mag,k|k-1} \right\|}$
    Normalize and Rotate Ang Vel k: ${}^{c}\boldsymbol{\omega}_k \leftarrow \mathbf{A}({}^{c}_{s}\mathbf{q}) \frac{{}^{s}\boldsymbol{\omega}_k}{\left\| {}^{s}\boldsymbol{\omega}_k \right\|}$
    Normalize and Rotate Ang Vel k-1: ${}^{c}\boldsymbol{\omega}_{k|k-1} \leftarrow \mathbf{A}({}^{c}_{s}\mathbf{q}) \frac{{}^{s}\boldsymbol{\omega}_{k|k-1}}{\left\| {}^{s}\boldsymbol{\omega}_{k|k-1} \right\|}$
  else
    Use Hardcoded Vectors for ${}^{c}\mathbf{v}_{sun,k}, {}^{c}\mathbf{v}_{mag,k}, {}^{c}\boldsymbol{\omega}_k \leftarrow [0\ 0\ 0]^T$
    Calculate Jacobian $\mathbf{H}_k \leftarrow \begin{bmatrix} 2\mathbf{S}({}^{c}\mathbf{v}_{sun,k|k-1}) & \mathbf{0}_{3x3} \\ 2\mathbf{S}({}^{c}\mathbf{v}_{mag,k|k-1}) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{1}_{3x3} \end{bmatrix}$
  Calculate Kalmal Gain $\mathbf{K}_k \leftarrow P_{k|k-1}\mathbf{H}^T_k(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}^T_k + \mathbf{R}_k)^{-1}$
  Update Error State $\delta\mathbf{x}_{k|k} \leftarrow \mathbf{K}_k(\mathbf{y}_k - \mathbf{y}_{k|k-1})$
  Expand quaternion ${}^{c}_{i}\mathbf{q}_{k|k} \leftarrow \left[\delta\mathbf{q}^T_{k|k} \ \sqrt{1 - \delta\mathbf{q}^T_{k|k} \cdot \delta\mathbf{q}_{k|k}} \right]^T \otimes {}^{c}_{i}\mathbf{q}_{k|k-1}$
  Calculate Full State $\mathbf{x}_{k|k} \leftarrow \left[{}^{c}_{i}\mathbf{q}^T_{k|k} \ ({}^{c}\boldsymbol{\omega}_{k|k-1} + \delta\boldsymbol{\omega}_{k|k})^T \right]^T$
  Update Posteriori Covariance $\mathbf{P}_{k|k} \leftarrow (\mathbf{1}_{6x6} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$
  Rotate Output $\leftarrow \left[({}^{c}_{i}\mathbf{q}_{k|k} \otimes {}^{c}_{s}\mathbf{q}^{-1})^T \ (\mathbf{A}({}^{c}_{s}\mathbf{q}^{-1}){}^{c}\boldsymbol{\omega}_{k|k-1})^T \right]^T$
  Increment $steps, t \leftarrow t + T_s$
  goto *Predict*.

3

AGENT-BASED FAULT DETECTION AND RECOVERY

"If you do not expect the unexpected, you will not recognize it when it arrives "

Heraclitus of Ephesus

Abstract

Failure detection, isolation, and recovery is an essential requirement of any space mission design. Several spacecraft components, especially sensors, are prone to performance deviation due to intrinsic physical effects. For that reason, innovative approaches for the treatment of faults of onboard sensors are necessary. This Chapter introduces and describes the concept of agent-based fault detection and recovery for sensors used in satellite's attitude determination and control subsystem. Its focuses on the implementation of an algorithm for addressing the linear drift in gyroscopes. The algorithm was implemented using an agent-based architecture to show the feasibility of this approach in the design of satellite's onboard software. It also presents, discusses and selects the most suitable strategy for fault detection and recovery required for the implementation of such software architecture.

Parts of this Chapter have been published in Acta Astronautica 139, 181-188 (2017)

Space missions involving the use of small satellites are becoming more popular due to their cost-effectiveness. Besides technology demonstration, there is an increasing number of Earth Observation missions requiring more demanding capabilities, as discussed by Belward and Skøien (2015). The need for a higher pointing accuracy is one of the most challenging problems to be faced in coming future, especially in small satellites. Missions like PROBA, described by Llorente et al. (2013), intend the demonstration of formation flying capabilities with critical dependency on spacecraft's performance. This situation increases the reliability constraints in satellites design and poses a risk for their operation. For that purpose, a more precise and reliable AOCS needs to be engineered; one that enables autonomous Fault Detection Isolation and Recovery (FDIR) features.

Two main approaches are typically used to design FDIR: hardware redundancy and analytical redundancy. Hwang et al. (2010) argue that analytical redundancy is more cost-effective and thus preferred over hardware redundancy in industrial applications, where physical redundancy is highly constrained. That is also the case for satellite systems, where it is preferable having software-based FDIR due to the lack of physical access to the system once it is deployed. Also, analytical redundancy methods provide more flexibility during the operations phase of the mission, since they can be adjusted to scenarios that were not accounted for during the design phase. In some cases, both physical and analytical redundancy can be combined to achieve higher reliability performance.

According to Yu and Jiang (2015) and Zolghadri (2012), within the analytical redundancy category, model-based methods have been most frequently used in fault detection for critical applications requiring Fault-Tolerant Control (FTC) due to their low implementation complexity. Model-based fault detection is centered in the generation of residuals, which act as fault signals. Residuals are generated from the comparison of the system measurements with their estimation values provided by a system model. A threshold function that can be either a constant or a variable can be used to calibrate the levels of detection depending on environmental noise. Additional information requires to be generated for fault isolation and recovery purposes. Examples of such applications are found in chemical processes by Ge et al. (2015) and automobile industry by Anwar and Niu (2014).

On the other hand, data-driven analysis has been gaining attention recently as presented by Khalastchi and Kalech (2018). That is becoming possible thanks to the advances in embedded computing technologies. Data-driven methods rely mainly on statistical data processing. Yin et al. (2012a) discus how data-driven methods are taking over model-based methods for isolation and recovery purposes due to their increased performance and implementations flexibility.

This dissertation proposes adopting an agent-based architectural style for the design of onboard software in satellites. This Chapter, specifically, looks to propose, discuss and to verify the implementation of FDIR algorithms in satellite's onboard software using an agent-based approach. To achieve this goal, first, it looks in depth at the methods used for FDIR in control systems. Then, it analyses how these functionalities can be implemented using an agent-based architecture. Finally, it verifies that the proposed FDIR implementation's strategy satisfies the performance requirements for a case study with AOCS components.

3.1. FDIR METHODS FOR CONTROL SYSTEMS

This section presents and discusses recent methods reported for fault and failure detection, isolation and recovery of sensors and actuators used in control systems. The primary objective of this Section is supporting the selection of fault detection and recovery algorithms for their implementation with agent-based software architectures. First, a distinction between faults and failures is made to avoid confusions. Faults are defined by Isermann (2006) as a deviation from the nominal behavior, while Failures are continuous interruptions of systems capabilities. This distinction is crucial since it must be taken into account when defining a strategy to address them.

According to the latest FDIR surveys by Hwang et al. (2010), Bittner et al. (2014b), Gao et al. (2015a) and Khalastchi and Kalech (2018) there are three main categories of fault diagnostic and recovery methods used in control system. These are model-based, signal-based and knowledge-based as depicted by the class diagram in Figure 3.1. Furthermore, combinations of these categories exist as hybrid methods.

Model-based methods require breaking down the systems into functions and processes that are formulated mathematically. A fault diagnostic observer checks the consistency between the physical measurements and the predicted output from the systems model. The observer can recognize deviations on its expected behavior using residual test algorithms, and then correct them by updating the configuration of systems parameters. Applications of model-based FDIR are presented by Jiang et al. (2008) and Marzat et al. (2012).

In signal-based methods, instead of using a specific input/output model, an observer extracts symptoms from the output measurements, and it compares them with a set of predefined fault signatures. Signal-based methods assume the input to be a reliable signal, and the observer verifies its consistency along the time.This is not the case for model-based methods where the trust is put in the model, rather than in the expected signature. Signal-based diagnostic methods are classified into three classes of algorithms: time-domain, frequency domain, and time-frequency combined.

Time-domain algorithms are preferred over the frequency-domain for continuous and dynamic processes since the interpretation of results is direct, which improves the response time. There are several applications in mechanical and electrical systems where time-domain methods are used for fault diagnostics. For example, in the work of Nandi et al. (2011), Dynamic Time Warping (DTW) is used as a signal-based algorithm for actuators to detects periodic impulse responses caused by faulty electromechanical components inside the system.

Within the frequency domain algorithms, Fourier Discrete Transformation is the most preferred. It is widely used in vibration analysis. Wang and McFadden (1993) present a work where faults are detected by measuring shifts on the frequency spectrum. Also, combining time and frequency domain methods is feasible for signal-based fault detection and isolation. The literature reports the use of the short-time Fourier transform and Wavelet transforms for motor eccentricity fault detection and isolation as described in He et al. (2015). The main concern of combining time and frequency approaches is their high computational load, which might not be suitable for resource constrained computers of small satellites running real-time fault detection and recovery algorithms.

The third FDIR category is the knowledge-based, which consist of two groups: quan-

titative and qualitative methods. The critical characteristic of knowledge-based methods is that they are data-driven. They include the use of artificial intelligence, expert systems, machine learning, pattern recognition and their variants, as described by Gao et al. (2015a). From a data-driven perspective, the FDIR problem can be divided into three phases: fault classification, regression, and reconfiguration.

Data-driven quantitative algorithms are subsequently divided into statistical, non-statistical and joint methods. According to Gao et al. (2015a) the statistical methods consist of Principal Component Analysis (PCA), Independent Component Analysis (ICA), Partial Least Squares (PLS) and Support Vector Machine (SVM). Most of them require large amounts of training data to capture key features of the process to improve their performance. The work of Samara et al. (2008) shows the application of statistical methods in FDIR for aircraft sensors.

Independent Component Analysis has been used to implement fault, identification, and reconstruction in three-axis gyroscopes, as presented by Li et al. (2011). SVM observers have been used in FDIR for re-configurable manipulators as described by Bo et al. (2011). One of the discussed benefits of using support vector machines is its strong ability to approximate nonlinear functions. However, it requires a high amount of calibration data, which make it less attractive for onboard software implementation. Within non-statistical algorithms, the Fuzzy Logic (FL) and Artificial Neural Networks (ANN) are popular. One of their main drawbacks for onboard software applications is the high amount of computing resources required for their execution.

Finally, the various methods from above can be combined to improve FDIR performance. For example, developing a fault detection observer with a model-based method, and a recovering algorithm using a data-driven algorithm can increase the FDIR efficiency without impacting the implementation's complexity.

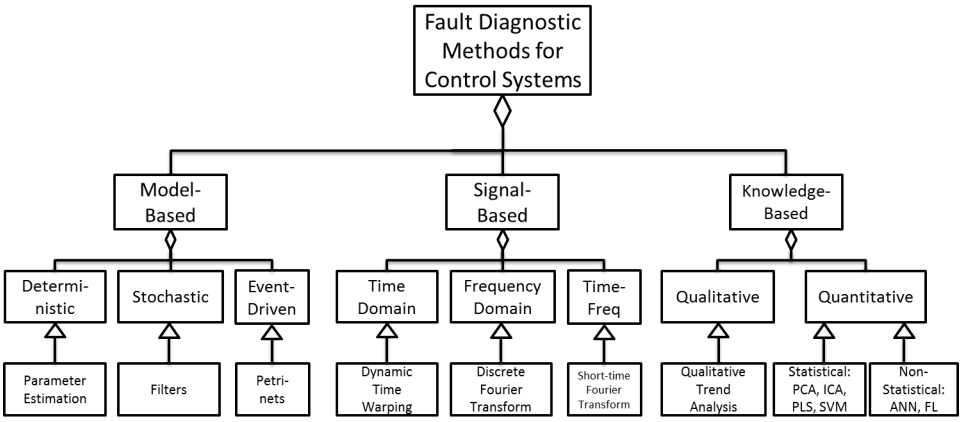


Figure 3.1: A Class Diagram for Fault Diagnostic Methods reported in literature for Control Systems

3.2. AGENT-BASED ARCHITECTURE FOR FDIR

According to Gao et al. (2015a), the functionalities needed for FDIR implementation are two: Fault Detection and Identification (FDI) and Fault Isolation and Recovery (FIR). FDI requires fault analysis and classification, while FIR preserves the system's integrity by isolating faulty components and executing recovery procedures when needed. Design of such algorithms is an important aspect of the process of space missions development. The more FDIR capabilities added, the more complexity that has to be managed during the OBSW implementations phase. From the analysis of the characteristics of the methods surveyed in Section 3.1 and the requirements of AOCS applications, model-based was the selected approach for FDI, whereas for FIR a knowledge-based method was identified as suitable for its implementation due to its operational flexibility.

In agent-based software architectures, both FDI and FIR functionalities can be assigned to specific agents, or they can be spread across all agents as internal behaviors. This Section aims to define the best strategy for their implementation within the onboard software of a satellite. For that purpose, a qualitative review of FDIR methods was performed in Section 3.1 to identify and select the most suitable methods to satisfy requirements and constraints of small satellites missions. Then, a trade-off analysis is proposed to identify the best strategy for FDIR implementation in AOCS computers running agent-based software. Finally, the algorithm is described in detail and validated numerically using AOCS operation scenarios for small satellite missions in Section 3.3

The process of strategy selection for FDIR implementation with an agent-based OBSW architecture is developed in three parts. Firstly, the options for FDIR are presented and discussed. Secondly, a set of trade-off criteria and their importance ratings (weights) are identified. These criteria are linked to the top level requirements defined in Subsection 1.7.1, more specifically to OBSW-ARCH-01, OBSW-ARCH-02, and OBSW-ARCH-04. Finally, the criteria are used to evaluate three implementation strategies with the Pugh Matrix method. As a result, one of the strategies is selected for the FDIR algorithm design and implementation.

3.2.1. FDIR IMPLEMENTATION OPTIONS

Figure 3.2 proposes a design option tree for FDIR implementation with an agent-based architecture. For the trade-off analysis several options are evaluated: one in which the FDI and FIR functionalities are centralized to a single agent. The second in which the functionalities are fully distributed among all the agents in the OBSW architecture, and a third where a combined approach is considered.

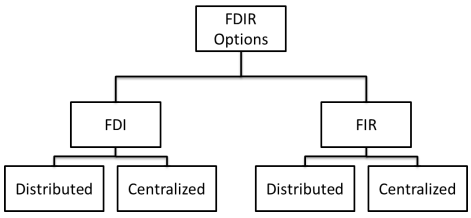


Figure 3.2: FDIR implementations options grouped by functionality and execution strategy

The main advantage of having a fully centralized implementation of FDI and FIR is that it simplifies the maintainability for extension the FDIR capabilities; In the other hand, a centralized approach makes the system vulnerable to a single point of failure, which is undesirable in Fault-Tolerant Control systems.

In a fully distributed FDI and FIR architecture, the response time of fault detection and correction gets improved given that each agent is capable of detecting correcting the errors locally. However, there is an overhead in systems internal communication, as well as an impact on the maintainability of the FDIR functionalities. That makes this approach less attractive for highly resource-constrained systems. The work of Katzela et al. (1995) compares the centralized vs. the distributed performance with respect to the computational effort needed to implement each of these approaches. It proved that the decentralized approach generally has considerably less complexity than the centralized approach, which helps to reduce the development cost.

The best way to balance performance and reliability of an agent-based FDIR architecture is the adoption of a combined strategy for its implementation. That means keeping one of the functionalities centralized to a few agents and distributing the other to all the software agents within the system. In that scenario, it is more convenient having a short response time in the detection and centralizing the response that requires further processing capabilities. For that reason, one of the options evaluated is having a distributed FDI and a centralized FIR.

3.2.2. TRADE-OFF CRITERIA

Four trade-off criteria are identified for the FDIR strategy selection. These are response time, communication overhead, resilience and maintainability.

Response Time: It is defined as the elapsed time since the fault is detected until the end of the recovery or reconfiguration protocol in the AOCS system. For FDIR its is critical to minimize the response time, at least in the detection and isolation phase. Low response time for these functions reduces the chances of failure propagation to other components within the system. According to Sherwood et al. (2001) response time also plays a significant role in the planning of activities for autonomous OBSW, where activity scheduling is deemed a continuous process.

Communication Overhead: This metric characterize the amount of communication between agents that is required by the FDI and FIR functions. Due to the distributed nature of MAS-based software, internal communication becomes a bottleneck that must be minimized to avoid the risk of network contention as discussed by Sonnek et al. (2010). For that reason, in case of fault or failure, if there is an increase in communication that saturates the data bus the system can lose its ability to recover. It has to be assessed when designing and implementing FDIR schemes with an agent-based approach.

Resilience: This aspect focuses on the ability of agents to resist software malfunctions during operations. For example, if an agent is in charge of fault detection and the agent itself fails, the system can redeploy the FDIR function to another agent as soon as the error is identified. According to Sayed et al. (2014) resilience is a measurement of robustness of distributed systems such as MAS-based software. That is the driver to consider it in the design of agent-based software architectures.

Maintainability: This criterion considers how difficult is the integration of new fault

mechanisms into the FDIR implementation. Maintainability is a concern for highly scalable systems. Garcia et al. (2004) find maintainability as one of the enablers for separation of concerns in distributed systems, hence its importance in the selection of a FDIR strategy that preserves this characteristic for MAS-based software.

3.2.3. TRADE-OFF ANALYSIS

The proposed architecture splits the FDIR functions into FDI and FIR. Then, according to the design option tree presented in Figure 3.2, four feasible implementations options are evaluated according the trade-off criteria. The implementation options are labeled A,B, C and D in the Table 3.1.

Each evaluation criterion is weighted on a scale from one to eight, being one the least important, and eight the most significant value for OBSW operations. For this trade-off, the most important criterion is deemed to be the software resilience due to the robustness requirements of satellite systems, while the least important is identified as maintainability because its impact on satellite operations is lesser than the others. The scores for weighting factor are shown in Table 3.1.

For each implementations option, the criteria are evaluated to have a positive effect (1), no effect (0) or negative effect (-1). Then, these values are used in the results row with their respective weight to compute a score for each implementation option. For instance, for option A the response time is deemed to have a positive effect since the same agent is able to detect and correct a potential fault in the system, compared to a fully distributed approach in option B where the response time is increased due to the extra communication time of the FDI and FIR agents. Options C and D balance this out looking for an intermediate solution.

The results from the trade-off show that worst option possible is having a fully distributed FDI and FIR architecture since response time, communication overhead and maintainability are negatively affected. Even when most of the criteria are achieved, the overall score is the lowest due to its lower performance compared to the rest. The second worst is the fully centralized FDI and FIR approach, where resilience is the primary concern, and therefore its overall score is also reduced. That makes the hybrid FDIR implementation strategy the way to balance both performance and robustness.

Options C and D have the same scores for response time, communication overhead, and resilience. What makes the difference between these two strategies has to do with the fact that according to the literature reviewed in Section 3.1, FDI methods are less complex to implement and maintain than FIR methods. That impacts the maintainability criteria for option C and therefore, makes option D the best option with the highest score. It implies that best strategy to implement FDIR with an agent-based approach is implementing FDI distributed across all the software agents and having the FDI centralized to few agents within the onboard software.

Based on the analysis of the characteristics for different FDIR methods used in control systems, the FDI function is implemented using model-based methods, focusing on residual testing algorithms, whereas the FIR function is implemented using a data-driven method with Independent Component Analysis. The main motivation for this selection is complexity of implementation and flexibility of calibration of the algorithm during satellites operations.

Table 3.1: Pugh Matrix for FDIR Strategy Selection

		FDIR Implementation Options			
Criteria	**Weighting Factor**	**(A) Fully Centralized FDI and FIR**	**(B) Fully Distributed FDI and FIR**	**(C) Centralized FDI and Distributed FIR**	**(D) Distributed FDI and Centralized FIR**
1. Response Time	6	1	-1	0	0
2. Communication Overhead	4	1	-1	0	0
3. Resilience	8	-1	1	1	1
4. Maintainability	2	1	-1	-1	0
	Sum(+)	3	1	1	1
	Sum(0)	0	0	2	3
	Sum(-)	1	3	1	0
	Result	**4**	**-4**	**6**	**8**

Legend:
1 Positive effect
0 No effect
-1 Negative effect

Weighting Factor:
1 | Least important
| |
8 | Most Important

Figure 3.3 summarizes the results of the trade-off analysis for the implementation of a fault detection and recovery strategy with agent-based software architectures. It links the implementation strategy with the respective FDIR methods selected for FDI and FIR functions within the software.

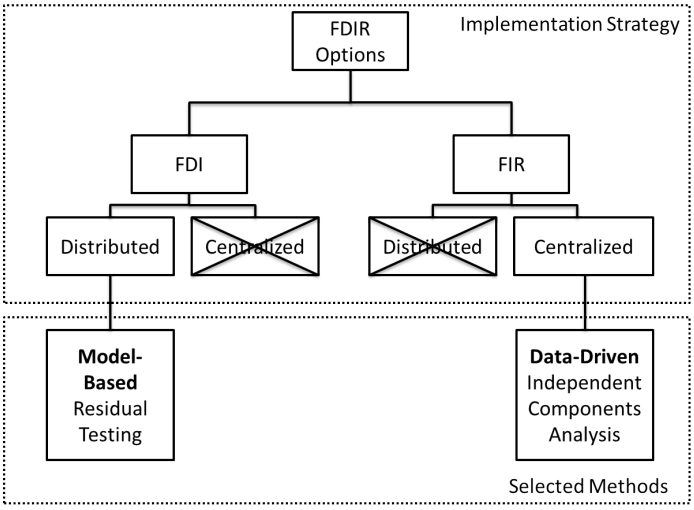


Figure 3.3: FDIR Implementation strategy and their associated methods for agent-based software architectures

The following Section focuses on demonstrating the application of the proposed FDIR architecture to small satellites systems. Particularly, on the implementation of the selected fault detection and recovery methods as a case study for the AOCS subsystem. Two operations scenarios are considered for numerical simulation with the purpose of showing the feasibility of a hybrid distributed-detection and centralized-recovery strategy using an agent-based approach.

3.3. AOCS CASE STUDY

This case study illustrates the application of the proposed FDIR implementation strategy to deal with errors in gyroscopes of an AOCS used in satellites subsystems. Firstly, it introduces the systems and components models. Then, it describes the implementation of the proposed FDIR architecture, as well as, its numerical validation with small satellite operation scenarios. Finally, results and potential improvements are presented and discussed.

3.3.1. SYSTEM MODEL

Consider the AOCS of a small satellite mission as depicted in Figure 3.4. It consists of two groups of blocks describing the plant model (external environment is colored gray), and the AOCS onboard software which includes the sensor's FDIR algorithm, the estimation and, the control algorithm. The plant model is described mathematically as a linearized and time-invariant state-space model presented as in the work of Gao et al. (2015b) using the following expressions

$$\begin{aligned} \mathbf{x}(k+1) &= (\mathbf{A}+\triangle\mathbf{A})\mathbf{x}(k)+(\mathbf{B}+\triangle\mathbf{B})\mathbf{u}(k)+\mathbf{B}_d\mathbf{d}(k)+\mathbf{B}_a\mathbf{f}_A(k)++\mathbf{D}_w\mathbf{w}(k) \\ \mathbf{y}(k) &= (\mathbf{C}+\triangle\mathbf{C})\mathbf{x}(k)+\mathbf{D}_s\mathbf{f}_s(k)+\mathbf{D}_v\mathbf{v}(k) \end{aligned} \tag{3.1}$$

where, $\mathbf{x}(k) \in \mathbb{R}^n$ is the system's state vector, $\mathbf{u}(k) \in \mathbb{R}^c$ is the control input vector, $\mathbf{y}(k) \in \mathbb{R}^m$ is the measured output vector, $\mathbf{f}_A(k) \in \mathbb{R}^a$ are the unexpected actuator faults, $\mathbf{f}_s(k) \in \mathbb{R}^s$ are the additive sensor faults, $\mathbf{d}(k) \in \mathbb{R}^d$ are the system's disturbances, $\mathbf{w}(k) \in \mathbb{R}^r$ is used for the measurement noise, $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{B}_a\mathbf{B}_d, \mathbf{D}_w, \mathbf{D}_s$ are known parameter matrices with the proper dimensions, and $\triangle\mathbf{A}, \triangle\mathbf{B}, \triangle\mathbf{C}$ are unknown modeling parameter errors including multiplicative errors.

For the purpose of this case study, the general model is simplified to focus on addressing sensor's faults, where the actuator faults ($\mathbf{f}_A(k)$), the disturbances ($\mathbf{d}(k)$) and the multiplicative errors ($\triangle\mathbf{A}, \triangle\mathbf{B}, \triangle\mathbf{C}$) are neglected.

The attitude is expressed using the quaternion notation $\mathbf{q} = [q_1\ q_2\ q_3\ q_4]^T$ which is obtained applying the equation of kinematics for satellites shown in (3.2) as

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \tag{3.2}$$

where, $\boldsymbol{\Omega}(\boldsymbol{\omega})$ is defined as

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}. \tag{3.3}$$

The spacecraft's dynamics are described by the derivative of its angular momentum as a function of its angular velocity $\boldsymbol{\omega}$ according to Euler's Equation described by Sidi (1997) as follows

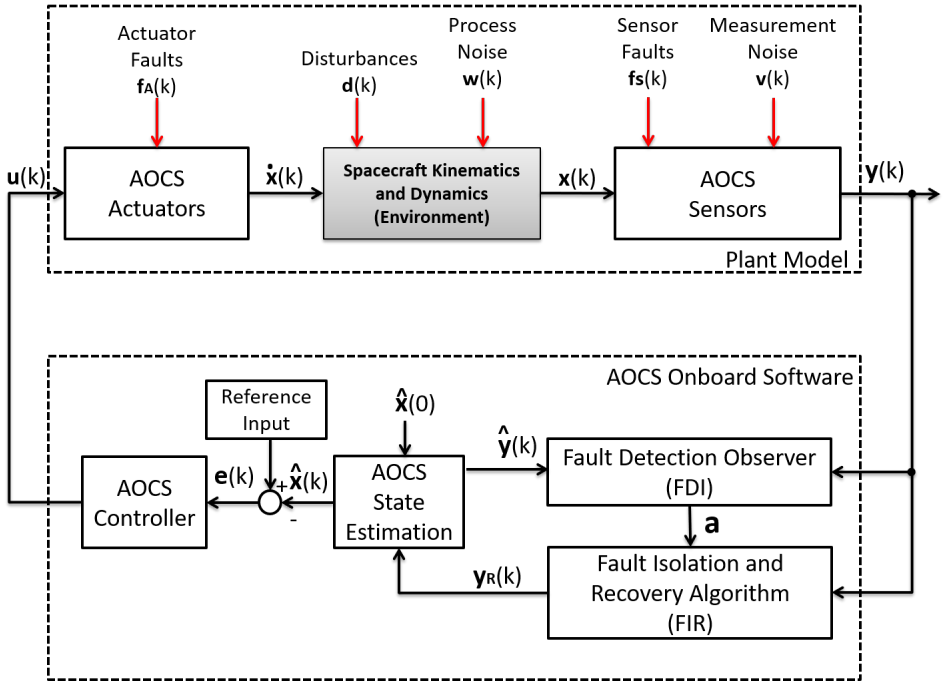


Figure 3.4: Block Diagram for the AOCS Case Study

$$\mathbf{I}_{SC}\dot{\boldsymbol{\omega}} = \mathbf{T} - \boldsymbol{\omega} \times (\mathbf{I}_{SC}\,\boldsymbol{\omega}) \tag{3.4}$$

where, $\mathbf{I}_{SC}$ is the spacecraft's moment of inertia matrix, $\boldsymbol{\omega}$ the spacecraft's angular velocity, and $\mathbf{T}$ denotes the addition of the control $\mathbf{N}_{Cont}$ torque and the disturbances torque $\mathbf{N}_{Dist}$ affecting the spacecraft (external torques). For example, the the actuation of reaction wheels and aerodynamic drag.

The expression in (3.4) can be reorganized to become

$$\dot{\boldsymbol{\omega}} = \mathbf{I}_{SC}^{-1}[-\mathbf{S}(\omega)(\mathbf{I}_{SC}\boldsymbol{\omega}) + \mathbf{N}_{Cont} + \mathbf{N}_{Dist}] \tag{3.5}$$

where, $\mathbf{S}(\omega)$ is a symmetric skew matrix described as

$$\boldsymbol{S}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & -\omega_x & 0 \end{bmatrix}. \tag{3.6}$$

Combining the kinematic and the dynamic differential equations in (3.2) and (3.6), the satellite's motion can be described by the state vector $\mathbf{x} = [q_1\ q_2\ q_3\ q_4\ \omega_x\ \omega_y\ \omega_z]^T$.

It is important to remark the dependency of both the kinematic and dynamic models on the angular velocity vector $\boldsymbol{\omega}$ for determining the orientation of the satellite. That

is the main motivation for putting special attention on mitigating gyroscope's measurement errors with the intention of making attitude estimation more precise and reliable. The following subsections will describe how to model gyroscopes measurements and their fault mechanisms.

3.3.2. GYROSCOPE MEASUREMENT MODEL

In general, the measured angular velocity $\boldsymbol{\omega}_m$ can be modeled as described by Bekkeng (2009) and Pirmoradi et al. (2009) as

$$\boldsymbol{\omega}_m = \boldsymbol{\omega} + \boldsymbol{\omega}_d + \mathbf{b} + \boldsymbol{\eta}_m. \tag{3.7}$$

In (3.7), $\boldsymbol{\omega}_m$ is the gyroscope output and $\boldsymbol{\omega}$ is the true angular velocity of the satellite in its local navigation frame. The model of $\boldsymbol{\omega}_m$ neglects errors due to sensor misalignment and scale factors included in the installation matrix of (2.19). Gyroscope's drift $\boldsymbol{\omega}_d$ is assumed to vary as a function of time. Precisely, as a linear-time function with parameters defined during the calibration of the gyroscope on the ground.

The environmental noise is added to the sensor measurement as a zero-mean white Gaussian noise, represented by the probability distribution function $\boldsymbol{\eta}_m$. The random walk is neglected for this case study as well as the rate walk, since both are assumed to be compensated during the design phase, and this case study focuses on errors during satellites operations.

The bias term $\mathbf{b}$ is estimated separately during the calibration phase using, for instance, the Allan Variance Method as discussed in the work of El-Sheimy et al. (2008) and, it is included as a constant term in the measurement model. $\boldsymbol{\omega}_d$ is not included in the state vector to reduce the computational burden of the state estimation algorithm. It is compensated by the FDIR algorithm described later.

3.3.3. GYROSCOPE FAULT MODELING

Faults in gyroscopes can be classified from different perspectives. For instance, from their form they could be systematic (e.g drift, bias, delay, saturation, stuck at) or random (e.g noise, walk), and from their time behavior, they could be permanent, transient or intermittent.

Table 3.2 summarizes the fault mechanisms and their modeling technique for a rate gyroscope sensor. This case study focuses on detecting and recovering from drift, rather than dealing with the other fault mechanisms.

Table 3.2: Gyro's fault mechanisms and associated modeling techniques

Fault mechanism	Form	Time behavior	Modeling technique
Drift	Systematic	Permanent	Linear time function
Misalignment	Systematic	Permanent	Installation Matrix
Bias	Systematic	Permanent	Constant
Walk	Random	Intermittent	Constant
Noise	Random	Intermittent	Probability distribution

The following subsections focus on the implementation description of a fault detection and recovery algorithm to address drift in gyroscopes using the agent-based software architecture selected in Section 3.2

3.3.4. GYROSCOPE INSTALLATION

Several Inertial Measurement Units are constructed using four rate gyroscope sensors in "3o+1s" configuration for measuring 3-axis angular velocity $\boldsymbol{\omega} = [\omega_x\ \omega_y\ \omega_z]^T$ as shown in Figure 3.5.

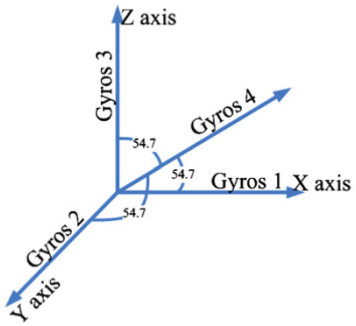


Figure 3.5: Gyroscopes Installation in a "3o+1s" Configuration as described by Li et al. (2011)

From the "3o+1s" gyros configuration, the angular velocity vector $\boldsymbol{\omega}$ for the spacecraft can be obtained as

$$\begin{bmatrix} Gyros_1 \\ Gyros_2 \\ Gyros_3 \\ Gyros_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{3.8}$$

Neglecting the misalignment faults during the installation process. Also assuming that the bias and the noise terms are already compensated ($\mathbf{b} = 0$ and $\boldsymbol{\eta}_m = 0$). The expression in (3.8) can be augmented to match the measurement model in (3.7) for including the drift value and a fault signal vector $\mathbf{a}$ as follows

$$\boldsymbol{\omega}_m = \begin{bmatrix} Gyros_1 \\ Gyros_2 \\ Gyros_3 \\ Gyros_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & a_4 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ \omega_d \end{bmatrix} \tag{3.9}$$

where, ω_d is a scalar value that accounts for the drift of the faulty gyro indicated by the fault vector $\mathbf{a}$ in a "3o+1s" IMU. For simplification purposes of this case study, it is assumed that only one gyro sensor is faulty at the time, so that the FDIR algorithm can estimate the drift and compensate it in real time.

3.3.5. FAULT DETECTION AND IDENTIFICATION ALGORITHM

According to the literature reviewed, and the FDIR strategy selected for an agent-based implementation in Section 3.2, the estimation agent in the AOCS software shall enable a behavior for model-based FDI. Chen and Patton (1999) discuss that performance features can be extracted from sensors or any other process and then they can be tagged into faulty or healthy using classifier algorithms. There are several strategies for generating such features, for instance: parameter estimation, state observers, output signal observer, and parity equations. One advantage of using state observers is that they are already part of the attitude and orbit estimation algorithm, which allows code reuse that lowers the computational overhead. Thus, the output from the AOCS state estimator can be compared to the output from the gyro sensor and the resulting difference $r(k)$ can be used to decide on whether or not a fault is occurring as

$$\mathbf{r}(k) = \mathbf{y}(k) - \hat{\mathbf{y}}(k). \tag{3.10}$$

In (3.10), the residual vector is composed of the difference between the AOCS sensors measurements $\mathbf{y}(k)$ and their estimated value $\hat{\mathbf{y}}(k)$. Using this residual as an input to an statistical classifier, a fault for the i^{th} IMU $i \in 1,2,3,...,N$ and its j^{th} gyro sensor $j \in 1,2,3,4$ can be declared using multiple hypothesis testing against an expected average measurement value μ_{ij} defined during IMU's calibration on the ground. The absolute value of the difference between the residual $r_{ij}(k)$ and the expected average μ_{ij} is compared to a threshold limit ε_{ij} defined according manufacturing specifications for each gyroscope sensor in the IMU as

$$a_{ij}(k) = \begin{cases} 0 \text{ if } |r_{ij}(k) - \mu_{ij}| \leqslant \varepsilon_{ij}, & \Longrightarrow healthy\ (H_0) \\ 1 \text{ if } |r_{ij}(k) - \mu_{ij}| > \varepsilon_{ij}, & \Longrightarrow faulty\ (H_1) \end{cases} \tag{3.11}$$

If the null hypothesis H_0 is not rejected, there is a healthy condition. In contrast, once the hypothesis test rejects the null hypothesis, a fault must be declared, which triggers the recovery algorithm and update the position $a_{ij}(k) \in \{1,0\}$ in the fault signal vector $\mathbf{a}$, which has dimensions $4 \times N$ that represents the feature (gyro-axis) where the fault is triggered. For example, if $a_{13}(k) = 1$ means that the IMU 1, gyroscope sensor 3 is faulty at instant k. The fault classifier assumes that one and only one gyro can fail at the same time. Also assumes that all the gyro measurements are statistically independent. This assumption is crucial for the selected fault recovery method.

A potential enhancement of this algorithm can be implemented taking the residual $\mathbf{r}(k)$ and constructing a set of parameters, for example, Squared Prediction Error (SPE), or the Hotelling T^2 statistics described in the works of Yin et al. (2012b) and Gupta (2006). Then, a comparison of these parameters with a threshold value for each gyro can be made in the same way that in (3.11) to declare a fault in gyroscope sensors. For this case study the approach used is the one described by (3.11), since it does not require any computational overhead in the AOCS computer.

3.3.6. FAULT RECOVERY ALGORITHM

From the literature review presented in Section 3.1, Independent Component Analysis (ICA) was selected as the most suitable data-driven method for implementing the recovery algorithm, specifically drift extraction and compensation for the proposed satellite operations scenarios. ICA has proven to be convenient when working on blind source separation problems as discussed by Comon (1994). This is the case for gyroscopes where the true value and the drift component are combined as indicated in (3.9). ICA also assumes that the observed features or measurements are independent among them, and they represent a linear combination of the hidden true value of the sensor and a faulty signal, in this case the gyroscope drift.

The gyroscopes sensors used in this case study are assumed to show a linear dependency on time for both angular velocity and the drift function. This kind of behavior is observed mainly when satellite performs attitude maneuvers, for instance, detumbling or payload pointing as shown in the simulation scenarios.

The model in (3.12) illustrates how to couple the fault detection algorithm with the recovery algorithm in a "3o+1s" gyroscope configuration. This expression was obtained by adapting the work presented by Li et al. (2011) to fit the proposed FDIR strategy for the operation's scenarios considered during the AOCS case study.

$$\boldsymbol{\omega}_m^i = \begin{bmatrix} Gyros_{i1} \\ Gyros_{i2} \\ Gyros_{i3} \\ Gyros_{i4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a_{i1} \\ 0 & 1 & 0 & a_{i2} \\ 0 & 0 & 1 & a_{i3} \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & a_{i4} \end{bmatrix} \begin{bmatrix} \omega_{ix} \\ \omega_{iy} \\ \omega_{iz} \\ \omega_{dj} \end{bmatrix} \tag{3.12}$$

In (3.12), the feature vector $\boldsymbol{\omega}_m^i$ represents the readings from the i^{th} IMU in the system, and $a_{ij} \in \{0,1\}, j \in 1,2,3,4$ is the fault detection flag calculated from expression (3.11). From the ICA formulation a signal vector $\mathbf{s} = [\omega_{ix}, \omega_{iy}, \omega_{iz}, \omega_{dj}]^T$ is defined to hold both the true value for angular velocity $\boldsymbol{\omega}$ and the value of the drift ω_{dj} for the j^{th} gyroscope sensor in the i^{th} IMU. As mentioned before, it was assumed a single fault event, meaning that only one value of $a_{ij} = 1$ can be triggered at the same time. Then, using the ICA algorithm, the FIR algorithm is able to demix the true signal vector $\mathbf{s}$ by calculating a demix matrix $\mathbf{W}$, that satisfies the following relation:

$$\mathbf{s} = \mathbf{W}\boldsymbol{\omega}_m^i \tag{3.13}$$

In (3.13), a demix matrix $\mathbf{W}$ that maximizes the statistical independence of the discovered signals $\mathbf{s}$ is estimated using an open source implementation of ICA called **FastICA** and proposed by Hyvarinen (1999). **FastICA** is an iterative algorithm that:

1. Choose an initial value for $\mathbf{W}$.
2. Update calculation of $\mathbf{W}$.
3. Normalize $\mathbf{W}$.
4. if not converged go back to step 2

FastICA requires to define a parameter (ρ) for adjusting the over-fitting and establishing the converging criteria required in step 4.

For the FIR algorithm implementation, a **FastICA** module was provided with an input vector $\boldsymbol{\omega}_m^i$ containing the simulated measurement values of the gyroscope sensors in a "3o+1s" configuration. These values were normalized and zero-mean centered before their actual processing. The output from **FastICA** was a vector containing the estimated drift signal d calculated from the input feature set and the true value for the angular velocity. The true values were expected to have a random noise component. That noise is compensated in the AOCS estimator by the filtering algorithm (e.g Kalman Filter), and it was not considered within the ICA algorithm implementation.

3.3.7. AGENT-BASED FDIR IMPLEMENTATION

Following the presentation of the detection and the recovery algorithms, the next step is describing their implementation within the AOCS software architecture. This subsection focuses in setting out the implementation details of an agent-based architecture for gyroscope's drift detection and correction based on the selected FDIR strategy.

During the software design phase, the AOCS software was broken down into five agents working together as depicted in Figure 3.6. These agents are:

- The Measurement Agent in charge of handling the interface with physical sensors.
- The Attitude and Orbit Estimation Agent that includes the FDI function.
- The Fault Isolation and Recovery Agent (FIR).
- The Attitude and Orbit Control Agent that focus on the control algorithm.
- The Actuator Agent that handles the interface with the satellite's actuators.

For this work, only the Fault Detection agent (FDI) and Fault Isolation and Recovery agent (FIR) were implemented using MATLAB™. They allow showing the behavior for both fault detection and recovery algorithms described above. The Measurement Agent, the Attitude and Orbit Estimation Agent, as well as the Control Agent and the Actuator Agent, were left out of the simulation model to focus only in the performance of sensors.

Figure 3.6 also shows the interaction among AOCS agents for fault detection and correction over an operation cycle of the onboard software. In that diagram the AOCS estimation agent request a new gyroscope measurement $y(k) = \boldsymbol{\omega}_m^i$ to the AOCS measurement agent. That measurement is also sent to the FIR agent for further correction is required. The estimation agent integrates an additional behavior to implement the FDI function using the estimated value $\hat{\mathbf{y}}(k)$ and the current measurement $y(k)$.

If a fault is detected, then the fault signal vector $\mathbf{a}$ is updated and the fault detection algorithm is triggered. The corrected measurement $\mathbf{y}_R(k)$ is sent back to the estimation agent for updating its estimated state $\hat{\mathbf{x}}(k)$. Then the estimation agent sends the current estimated state vector to the control agent which will update the output $u(k)$ and define the commands required at the actuators. These commands are forwarded to the actuator agent for maneuver execution if needed. The reference value for the AOCS orientation is received from the CDHS and shall be updated before the control algorithm execution.

One aspect to highlight from the proposed agent implementation strategy is its modularity. Using agent-based architectures the software complexity is controlled, so that new fault detection and recovery algorithms can be tested without a significant impact on the control nor the estimation implementation, just by modifying the behaviors inside the agents.

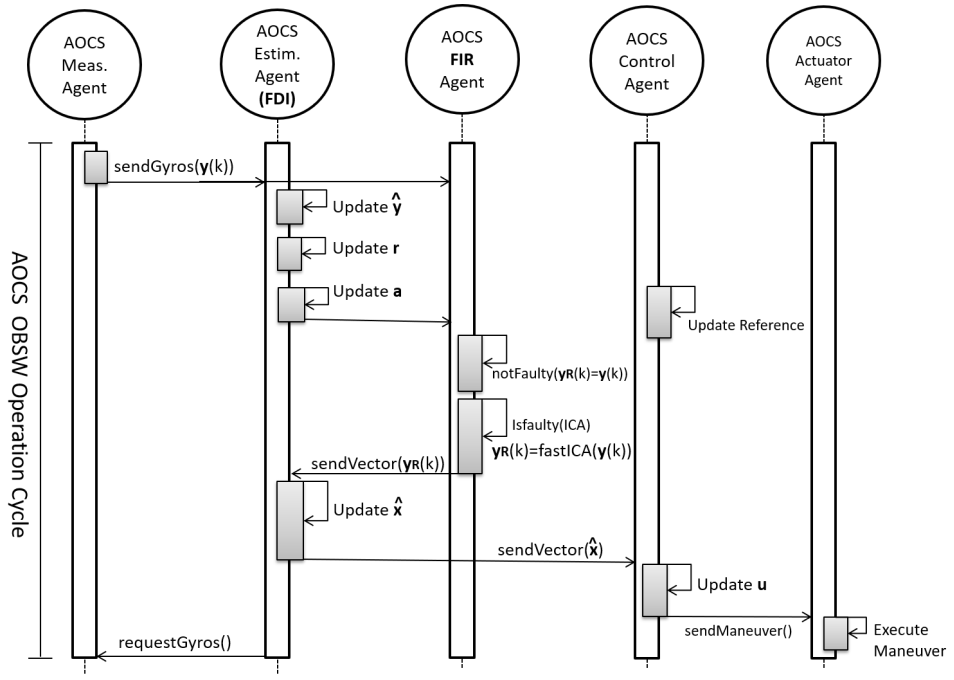


Figure 3.6: Data Flow Interaction for the proposed FDIR Scheme with an Agent-based Architecture

3.3.8. SIMULATION SCENARIOS

The simulation environment was developed using MATLAB™. There, the behavior of both fault detection and fault recovery agents were implemented to reproduce two spacecraft modes: detumbling and payload pointing. The input signals were synthesized for each scenario separately, using the sensor and fault model presented in subsection 3.3.3. In both operations scenarios, the angular velocity, as well as the drift, were assumed to behave linearly during the operation period. The reason for this assumption, is to have a reference function to compare the measurement values over time since they are needed for verification purposes.

Also for each scenario, both variables were characterized by sensor parameters extracted from the mission's design. The threshold used to declare a fault was determined based on the gyroscope's tolerance specifications reported for the missions.

The following subsections describe each operation's scenario implemented in the simulation environment. It focuses on verifying that once the fault is triggered, the FIR algorithm can correct the measurements received from the faulty gyroscope sensor.

SCENARIO 1: DETUMBLING MODE FOR THE "FLYING LAPTOP" MISSION

Detumbling is the first critical activity after the deployment of a satellite. It requires the satellite to follow a specific tumbling rate profile according to the mission design. Any drift in the measurements might cause delays in the commissioning of the satellites operations. If the tumbling rate is not achieved, the success of the mission is at stake. Therefore, it is crucial to complete the detumbling maneuver within a few orbits after satellites are deployed.

The "Flying laptop", described by Kuwahara et al. (2009), is a micro-satellite with a mass of 120 kg flying in a sun synchronous orbit of about 600 km altitude. The detumbling period requirement was established in the range of 6000 to 12000 seconds.

The Flying Laptop's ADCS was equipped with fiber optical gyroscopes with a drift rate specification of 3 °/h. The initial angular velocity was determined as 10 °/s, and the mission requirement was to stabilize it to 0°/s within a tolerance on 1 °/s as discussed by Grillmayer et al. (2006). For the proposed simulation scenario, it is assumed that detumbling shall be completed within 6000 s, which corresponds to one orbital period. Table 3.3 summarizes the simulation parameters considered for the Flying Laptop scenario.

Table 3.3: Simulation parameters used for the operation scenario with the Flying Laptop mission

Input Parameter	Description	Value [Unit]
InitRate	Initial tumbling after deployment	10 °/s
EndRate	Final tumbling rate for operations	0 °/s
Driftlimit	Maximum gyroscope drift accumulated in one orbit	5 °/s
SampleT	Sample period for measurements	1 s
OrbT	The period required to complete one orbit around Earth	6000 s
Meas_Noise_A	Measurement noise variance for amplitude	0.001 °/s
Meas_Bias	Gyroscope measurement bias	0 °/s
ϵ_{ij}	Drift threshold used to trigger a fault	0.5 °/s
μ_{ij}	Expected increase of the drift as a function of time	0.0001 °/s

The input signals vectors for true angular velocity $\boldsymbol{\omega}^i$, measured angular velocity $\boldsymbol{\omega}^i_m$ and drift input ω_{dj} were generated using the information from Table 3.3 and, plotted in Figure 3.7. The injected drift was generated considering the orbital period and the drift rate specification provided by the gyroscope's manufacturer, so that after 6000 s, the maximum drift expected was 5 °/s as shown in Figure 3.7 (right y-axis). The faulty angular velocity magnitude $|\boldsymbol{\omega}^i_m|$ was synthesized by adding the true angular velocity $\boldsymbol{\omega}^i$ plus the injected drift vector. It was evident that if the drift was not compensated, the detumbling period required to be extended, impacting early operations performance of the mission.

The fault detection behavior (FDI) within the Estimation Agent (see Figure 3.6), was provided with the faulty angular velocity measurement vector $\boldsymbol{\omega}^i_m$, that calculated the residual $\mathbf{r}(k)$ to perform the multiple hypothesis testing described by (3.11) . The trigger

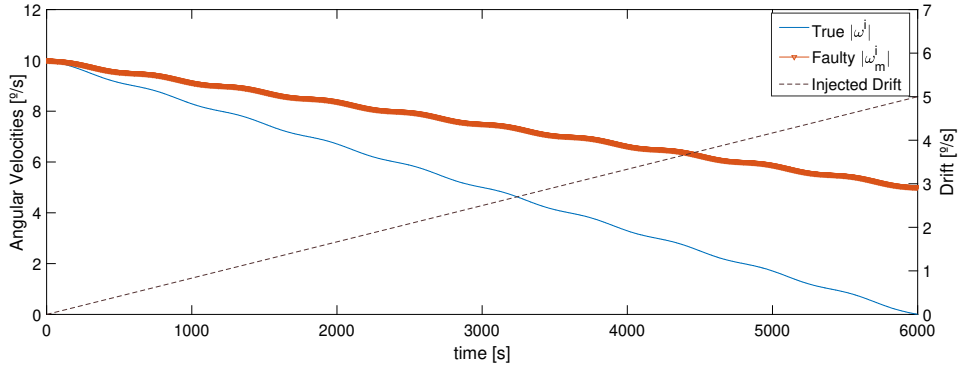


Figure 3.7: Input variables generated for the Flying Laptop simulation scenario including true angular velocity magnitude, Measured angular velocity magnitude and drift injected to the faulty gyroscope sensor

condition was determined by the comparison with the ϵ_{ij} value shown in the table, so that a faulty signal a_{ij} vector was generated as shown in Figure 3.8 (right y-axis). The a_{ij} vector determined the moment where the Fault Isolation and Recovery algorithm had to be triggered in the AOCS FIR Agent.

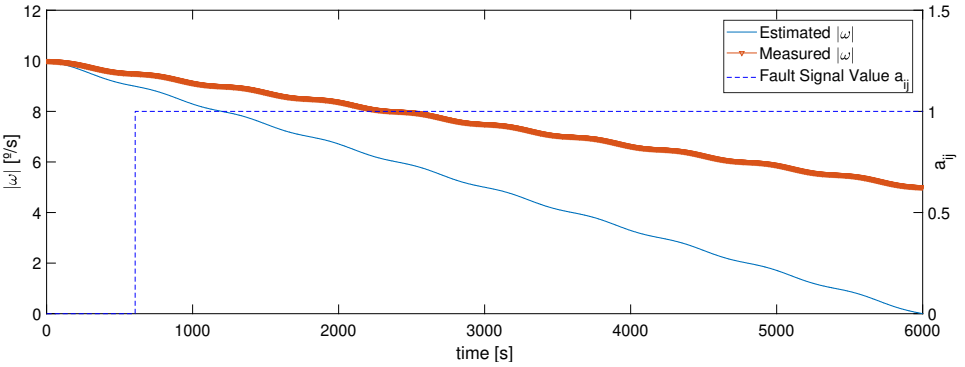


Figure 3.8: Results of the implementation for the Fault Detection Algorithm proposed in (3.11)

The **FastICA** algorithm running on the FIR agent was provided with gyro's measurements vector $\boldsymbol{\omega}^i_m$, but it required to have the data pre-processed because the **FastICA** implementation needed a normalized zero-mean centered input vector to work properly. That was achieved calculating and subtracting the mean value to the input vector. Additionally, a scale factor was defined to adjust the output magnitude. This parameter depended on the highest drift value expected at the end of the maneuver. In this case, it was about $5°/s$ that was the maximum drift value injected. The reason defining this scale factor was because the algorithm worked with normalized input data, so the output had to be adjusted to the input scale.

In Figure 3.9, the plot A shows the drift behavior at the fault recovery agent. For simplicity, it assumes that recovery algorithm is triggered at $t = 0$ s. The recovery algo-

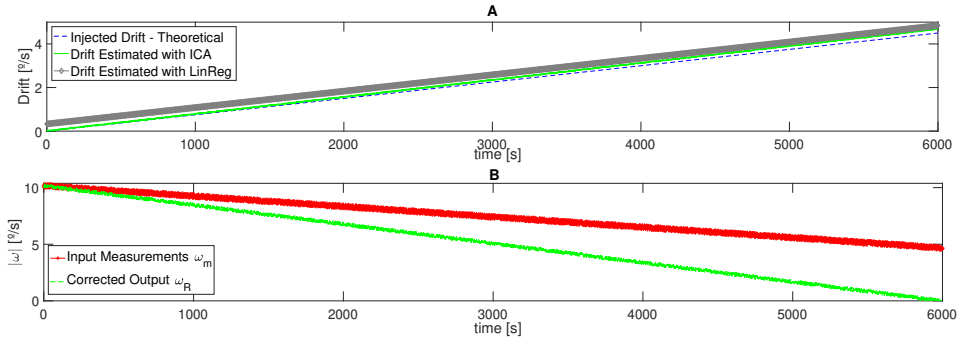


Figure 3.9: Simulation results of Flying laptop scenario for detumbling maneuver. (A) Drift Estimation performance for ICA and Linear regression method. (B) Input-Output for FastICA Implementation

rithm allowed to estimate the faulty sensor's drift and compensate it as shown in plot B where the faulty input was indicated as the thick-red line and the corrected output was shown as the thin-green line. In plot A the injected drift (dashed-blue line) was indicated for comparison purposes. The drift estimated using ICA (solid-green line) was compared to the theoretical drift injected, to show that algorithm is able to demix the drift coming from the faulty sensor. Additionally, a drift estimator implemented using a linear-regression model was included in the plot A (thick-gray line) with the purpose of comparing its performance and implementation's complexity to the ICA algorithm.

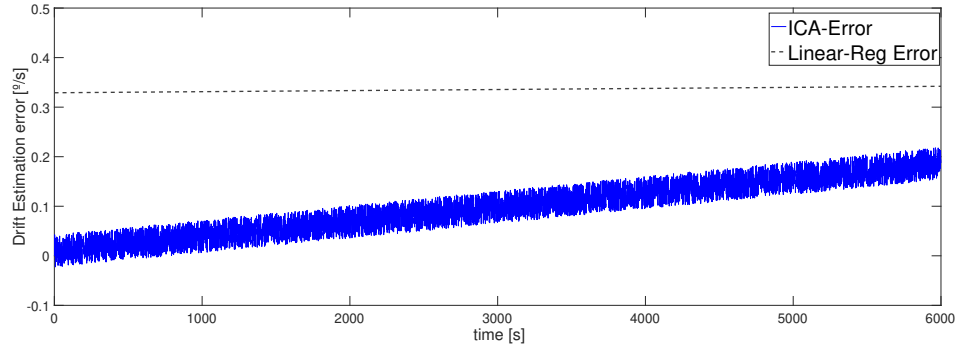


Figure 3.10: Comparison of two drift estimation algorithms for fault recovery of gyroscopes in the Flying Laptop operation scenario

Figure 3.10 shows the drift estimation error for the FastICA Implementation (solid-blue line) and a linear regression model (dashed-gray line). These errors were generated by subtracting the theoretical injected drift, so that the performance of both algorithm can be analyzed. It was clear that both algorithm overestimated the injected drift. However, the ICA implementation outperformed the linear regression model. In that sense, having a data-driven algorithm shows benefits in terms of performance. In the other hand, the linear-regression model shown a stable error over time, whereas the ICA implementation shown a deviation over time that can be a concern in the long-term.

SCENARIO 2: PAYLOAD POINTING MANEUVER FOR AEROCUBE-OCSD MISSION

In 2012, NASA selected the Aerospace Corporation to develop a technology demonstration mission to test COTS components in optical communications with CubeSats. The proposed concept included an optical payload using beam spreads in the range of miliradians. According to Janson and Welle (2014b), the mission's objective was to establish a communication link between the AeroCube-OCSD satellite and a telescope located in California, USA. For that purpose, the mission used a 10 W laser with 1.4° angular beamwidth on a 1.5 U CubeSat.

Figure 3.11 shows the angular rate profile and the range distance for AeroCube-OCSD on a zenith pass over the optical ground station. Based on data from Janson and Welle (2014a), the required magnitude of the rotation rate of the satellite was specified to be less than 0.8 °/s at 600 km altitude. The minimum elevation angle was determined as of 30 °, and a maximum laser communication period was defined to be 180 s. From the AeroCube-OCSD mission design, the maximum drift for the gyro was estimated to be 0.3 °/s during the communication period of 180 s when the satellite was in range.

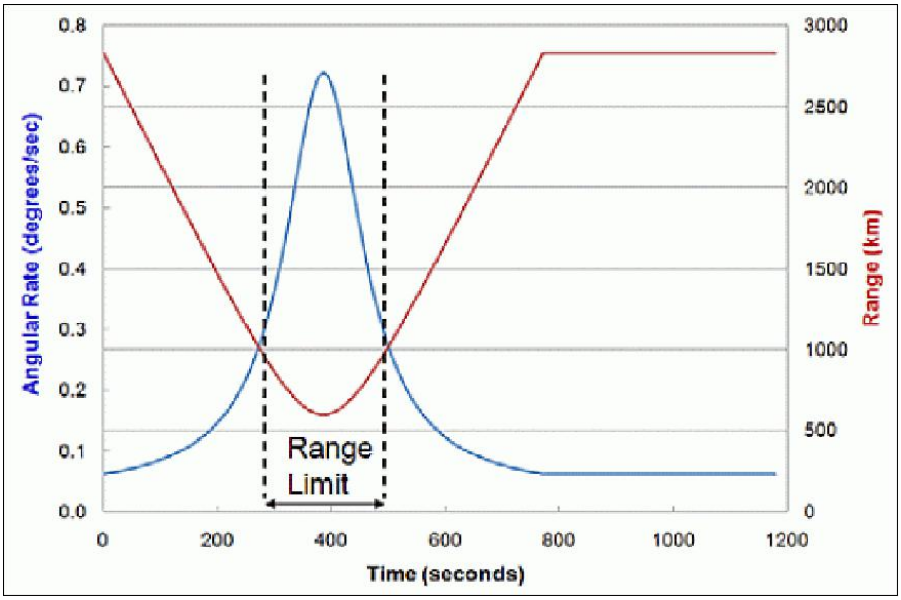


Figure 3.11: Angular velocity and range distance profiles for the payload pointing maneuver of AeroCube-OCSD mission by Janson and Welle (2014a)

Using above's information, a simulation scenario was created to emulate the payload pointing maneuver of AeroCube-OCSD satellite as shown in Table 3.4. The drift was assumed linear during the maneuver period. The effects of optical interference on the ground were neglected.

In the simulation setup, the time window for the maneuver was defined as 180 s. It was assumed that the angular rate profile started in 0.25 °/s and it linearly increased during 90 s to reach 0.75 °/s. Then, it linearly decreased again to 0.25 °/s at the end of the maneuver period. This behavior was intended to represent, to some extend, the angular

Table 3.4: Simulation parameters used for the operation scenario with the AeroCube-OCSD mission

Input Parameter	Description	Value [Unit]
InitEndRate	Initial-Final tumbling rate for payload pointing	0.25 °/s
MaxRate	Maximum tumbling rate for payload pointing	0.75 °/s
Driftlimit	Maximum gyroscope drift accumulated during the payload pointing	0.3 °/s
SampleT	Sample period for measurements	1 s
ManeuverT	The period required to complete the payload pointing maneuver	180 s
Meas_Noise_A	Measurement noise variance for amplitude	0.001 °/s
Meas_Bias	Gyroscope measurement bias	0 °/s

rate profile shown in Figure 3.11, since the function to reproduce it exactly was not available in the literature. The drift for the gyroscopes were determined to be nominal 0.3 °/s (3σ) in 10 minutes; Sensor noise was also added to the angular rate profile to make it consistent with the gyroscope measurement model, even when it was known that it was going to have an impact in the performance of the ICA algorithm.

Figure 3.12 shows the results of this simulated scenario reproducing the angular rate profile for AeroCube-OCSD when gyro's drift affected the sensor measurements.

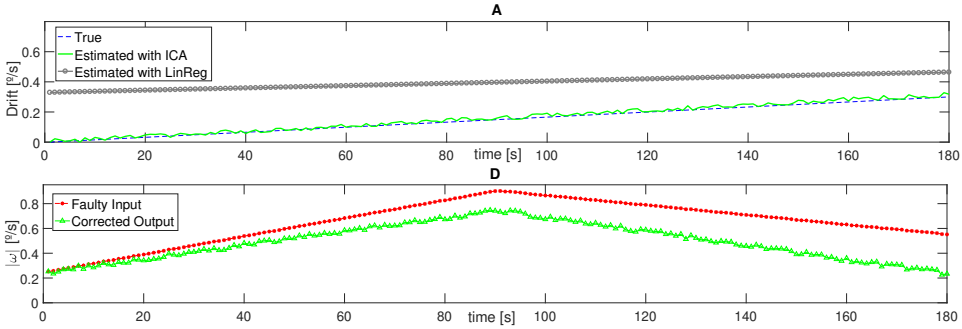


Figure 3.12: Simulation results of AeroCube-OCSD scenario for payload pointing maneuver. (A) Drift Estimation performance for ICA and Linear regression method. (D) Input-Output for FastICA Implementation

Subplot A shows the drift affecting the angular velocity measurements. Subplot A also shows the injected drift (dashed-blue line) , the estimated drift calculated by the FIR Agent when receiving the faulty signal vector (solid-green line), as well as the drift estimated using model-based approach with linear regression (dotted-gray line). Plot A shows that ICA has a better performance to estimate the injected drift compared to the linear regression model.

In Subplot D, the effect of drift in the angular velocity profile was plotted as the Faulty input (dotted-red line) with its corrected output (solid-green line) after the recovery procedure. In that plot, the faulty signal exceeded on 20% the rotation rate requirement during the maneuver, which did not allow the satellite to achieve the required pointing

accuracy on the pass over the ground station. Additionally, the fault caused an asymmetrical profile, increasing the required time window for the communication to 220 s (increase of 22%). This deviation had a direct effect on the maximum downlink/uplink speed that can be achieved during the pass, meaning that drift can decrease the performance of the satellite.

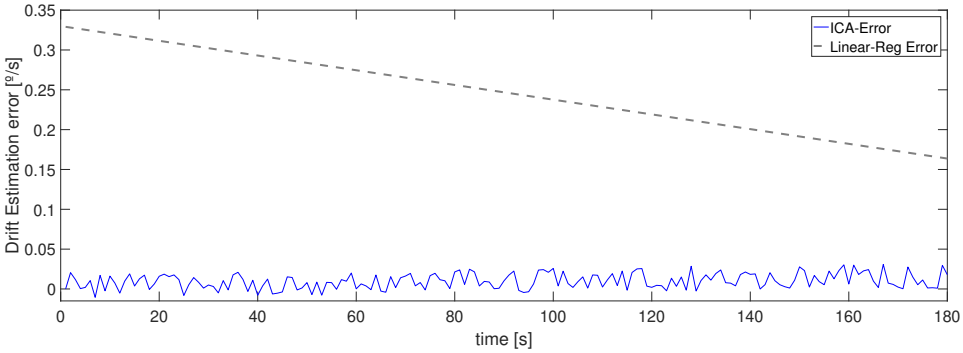


Figure 3.13: Comparison of two drift estimation algorithms for fault recovery of gyroscopes in the AeroCube-OCSD operation scenario

Figure 3.13 shows two interesting findings. Firstly, the magnitude of the error of the linear-regression model (dashed-gray line) was in average 25 times bigger than ICA (solid-blue line). The second element to note is that the error of the linear regression model was decreasing over time, and the error of ICA was slightly increasing, but mostly kept stable along the maneuver. Also, ICA shown more variability than the linear-regression model as product of the Gaussian noise injected on the input signal.

3.4. RESULTS ANALYSIS

This Section analyzes the proposed FDIR scheme from both the architectural and the performance of the methods selected for FDI and FIR functions. These analyses are based on the results from the AOCS simulation scenarios, and they are intended to draw conclusions and recommendations to software designers and space systems engineers.

Concerning the architecture for fault detection and recovery implementation, the use of a distributed FDI approach showed benefits regarding code reuse, resilience and balanced response time. That enabled a faster drift detection for the gyroscope operation scenarios implemented. Also, this type of architecture allowed the capabilities for the implementation of multiple observers without significant impact in the overall software complexity, which can be used to implement detection on multiple fault mechanisms besides drift. Related to the method selection, using a model-based algorithm for drift detection showed a good performance for both scenarios tested.

On the other hand, for the FIR function the results of the case study scenarios showed some challenges with the implementation of both the architecture and the recovery algorithm. It was clear that for this function the computational requirements were higher, therefore its implementation complexity; However, based on the results, there some aspects that need to be developed further. For example, from its implementation archi-

tecture, it is necessary to define a consensus strategy among FDI and FIR agents that guarantees resilience with a minimal impact in communication overhead.

Related to the implementation of **FastICA** algorithm for drift recovery some concerns require attention during onboard software development. For instance, in some cases, the algorithm took several iterations to converge to a solution for a data batch. Also, the need to have the input data pre-processed added computational cost. These two behaviors pose a risk for the onboard processing budget, which can impact workload performance negatively.

Three ambiguities on ICA were observed based on the results for the simulation scenarios. Firstly, if the input data was not normalized, then the algorithm was not able to determine the variances of the independent components. The reason of that, was because the matrix of signal and parameters were unknown. That was addressed normalizing the measurements input and re-scaling the output of the algorithm before compensating the drift. The second ambiguity was the sign of the calculated components, which had to be addressed by also post-processing the output. The third ambiguity observed was related to the order of calculated components in the output vector. That required defining a permutation matrix and adapting the input model to solve this matrix, which again caused computational overhead. An additional problem was observed when adding Gaussian noise to the input vector, which decreased the performance of ICA for the components separation and caused the algorithm to overestimate the drift in all cases.

Concerning the time stability, ICA shown that for long term maneuvers the error in estimation was increasing, so that it was comparable with error of the drift estimation method with linear regression model. For short-time maneuvers ICA clearly outperforms the model-based recovery method, even when adding noise.

3.5. CHAPTER SUMMARY

This Chapter has presented and discussed the results of an agent-based architecture for drift recovery in gyroscopes. A qualitative assessment for FDIR methods was carried out, including models-based, signal-based and knowledge-based methods. The selected method for FDI was a model-based technique, while the FIR was selected to be a data-driven method.

A trade-off analysis was carried out to identify and select a strategy for FDIR implementation using an agent-based architecture. This strategy suggests that the most efficient approach for implementing FDIR requires spreading the FDI across all the functional agents of the system, while having a few centralized agents to handle the FIR procedures.

Two simulation case studies with small satellite missions were implemented to demonstrate the algorithm feasibility and effectiveness. Numerical simulations showed a better performance of the proposed data-driven method with respect to a purely model based approach. The ICA algorithm used for drift estimation shown several implementation considerations that requires further research. These issues can limit the potential adoption of data-driven methods for fault recovery in onboard software design.

Future work includes the development of the agents for state estimation and attitude control, and its integration in other maneuver cases study scenarios.

4

MULTI-AGENT COMMUNICATION IN SATELLITE SOFTWARE

The most important thing in communication is hearing what isn't said.

Peter Ferdinand Drucker

Abstract

Communication is the most critical capability needed for the successful implementation of multi-agent based software in satellites systems. To enable that capability, spacecraft require to put in place distributed data buses featuring reliable protocols with built-in fault detection and recovery. This Chapter proposes and describes an architecture for multi-agent system communication that can be adopted in the design of fault-tolerant onboard software for satellites using an agent-based approach. For that purpose, some sections focuses on the message transport implementation with the Controller Area Network (CAN) protocol, so that synchronization is guaranteed by design. It also demonstrates the implementation of this architecture with high-throughput demanding scenarios with the ADCS case study. Finally, an algorithm for bus utilization balancing is proposed, so that the multi-agent based software performs according to specifications.

Parts of this chapter have been published in IEEE Transactions on Aerospace and Electronic Systems -TAES- pages 1014 - 1025, Volume: 56 , Issue: 2 , April, (2020) doi: 10.1109/TAES.2019.2940341

Multi-agent systems are described by Chopra et al. (2013) as dynamic organizations composed of autonomous software entities running over a distributed computing environment to achieve a common goal. Agents require interacting with each other to cooperate and negotiate while achieving their goals. According to Chen and Su (2003), communication is one of the core services that enable the development and implementation of MAS-based software. The best way of dealing with the implementation complexity of highly distributed systems is by introducing different abstraction levels in the design of their communication strategy. Li and Kokar (2013) argues that agent communication protocols are required to specify a minimum set of rules for agent communication within the multi-agent system boundaries. There is also a consensus on the importance of autonomy as part of the agent communication semantics.

The Foundations of Intelligent Physical Agents (FIPA) establishes the requirements for an agent communication language as a set of standards for multi-agent systems platforms. For example, the specification FIPA00067 in FIPA (2002b) describes the reference model for message transport, in which several communication requirements and constraints are documented, as well as their message structure.

The communication process of a MAS requires two main groups of functionalities. The functions that allow high-level interaction of agents known as the Agent Interaction Protocols (AIP), and the lower-level functions related to the information transport implementation in charge of the Message Transport Protocol (MTP). The AIP allows the representation model for an agent. That includes an ontology, a content language or any other taxonomy and vocabulary required to represent useful information according to the agent's semantics. These language primitives are integrated into the Agent Communication Language (ACL) specifications.

The Message Transport Protocol (MTP) provides application-specific rules for the implementation of messaging exchange. They enable the physical and logical link between agents. The reason MTP are application-specific, is because they depend on the application requirements and constraints as well as their implementation technologies. In some cases, they need to be optimized as discussed by Bravo et al. (2015).

In space-related applications, the use of multi-agent systems is deemed necessary when a mission requires advanced autonomy features during operations. For example, Long et al. (2005) discuss the use of multi-agent systems to improve the speed and precision of fault diagnostic methods using satellite telemetry data. One of the main challenges for implementing MAS-based telemetry collection is the need for a reliable communication channel between multiple subsystems and components of the spacecraft data bus.

This Chapter proposes and describes an architecture for agent communication in MAS-based software for satellites. First, it describes qualitative aspects related to the communication process in multi-agent systems, for instance, their agent's communication protocols and their message transport protocols. Then, it focuses on developing an analytical model to describe and characterize the workload of the spacecraft's distributed communication bus. This bus utilization model is validated using an AOCS case study with critical operation scenarios for small satellite missions.

4.1. AGENT COMMUNICATION LANGUAGES

The communication languages are the core of any multi-agent systems implementation. It is also necessary to use a reference model for describing multi-agents system platforms. FIPA establishes a reference communication architecture for that purpose. Figure 4.1 shows the elements that constitute the typical multi-agents system platform according to the FIPA specifications. Each of these elements can be modeled as a set of software objects (components) grouped into layers.

From the communication point of view, agents can interact with other agents within the same agent's platform or with agents on a different platform. The Message Transport System (MTS) provides communication capabilities to agents attached to a single agent platform using the Internal Platform Message Transport (IPMT), and the communication with agents living on a different platform using the Agent Communication Channel (ACC).

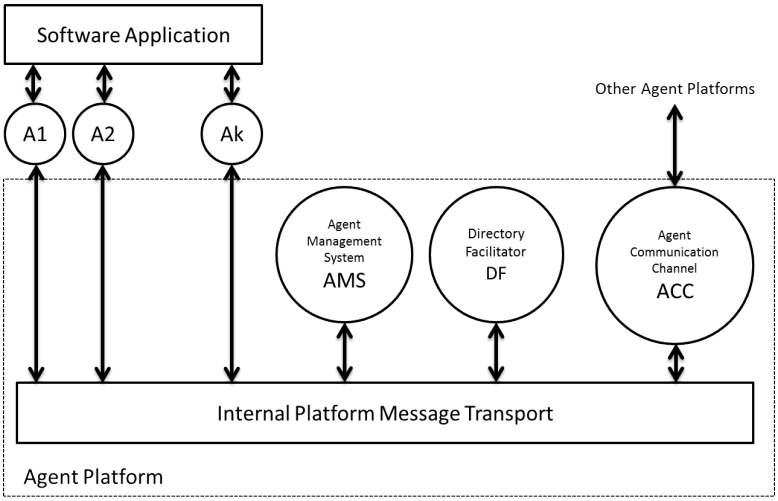


Figure 4.1: FIPA reference model for a multi-agent system platform.

The standard FIPA (2002a) specifies that the communication between agents is implemented by exchanging messages containing two parts: a message envelope and a message payload. The message envelope is required to express information needed for the message transport protocol, and the message payload is used to express the FIPA-ACL content. The ACC may add information, but it is not allowed to remove any content. The following subsection focuses on the communications architecture and protocols for agent's communication implementation.

Agent communication protocols are grouped into a layered structure to reduce the implementations complexity of communication by separation of concerns as discussed by Nguyen et al. (2011). Table 4.1 shows the layers and their respective functions in the agent communication stack. The ACL is highlighted as the most important layer for this thesis.

Table 4.1: Agent Communication Stack described by the standard FIPA (2002a)

Layer	Description
Conversation	Sequence of valid messages within specific operation context. It is defined at application level. It implements a set of agent interaction protocols to describe communication patterns.
Content Language	It is used to express the content of communication between agents. For that it uses grammar and lexical definitions to describe agents interactions. FIPA defines four types of content languages: FIPA-SL FIPA-CCL, FIPA-KIF and FIPA-RDF.
Agent Communication Language	The ACL specifies the syntax and semantics of messages that agents exchange. It pre-defines a collection of message types called performatives. FIPA-ACL and KQML are the most relevant and known.
Message Envelope	Defines the routing and encoding policies for the messages, and describes how to implement such policies. For example XML-based envelopes.
Message Transport Protocol	Defines the structure of a ACL message using standardized objects to information exchange. For example: IIOP, CORBA, RMI, HTTP
Data Transport Protocol	Establishes the data transport protocols required for agents messages to transfer their information. For example, TCP or UDP.
Data signalling	Defines the physical mechanisms for data propagation within the communication network. For instance, Ethernet, 802.11, or CAN

4.1.1. AGENT INTERACTION PROTOCOLS

Knowledge Query Manipulation Language (KQML) described by Finin et al. (1994) and FIPA-ACL described by Fipa (2002) are the most common communication languages used in multi-agent systems. They are intended for three purposes which are: defining the message structure, describing the communication acts between agents, and enabling the communication protocols required for supporting the inter-operation of multiple agents within the same execution environment.

The FIPA-ACL is oriented to standardize the encoding, semantics and pragmatics of messages in multi-agent-based applications as described by Fipa (2002). It does not include any requirements on the internal transportation of messages, which is left for application-specific implementation. Fipa (2002) establishes that a FIPA-ACL message can contain one or several predetermined parameters including the type of communicative act, sender, receiver, content, ontology among several others defined in the specification. Additional user-defined parameters can be added to the message structure, but they are not specified in the standard. For standardizing interaction between agents, FIPA-ACL defines and describes a set of interaction protocols summarized in Table 4.2.

Table 4.2: Summary of Main FIPA-ACL Agent Interaction Protocols

AIP Name	**FIPA Document Number**	**Protocol Description**
Request	SC00026	This protocol allows one agent to request another to perform some action, which might be able either to refuse or agree with it. If the receiver agrees, it might complete the task and inform the result or that it is done, if it fails to complete the task the agent shall send a failure message to the requester agent.
Query	SC00027	An agent is able to request to perform an action on another agent.
Request-When	SC00028	Agents are allowed to request an receiver agent to perform an action at the time of a given precondition becomes true.
Contract-Net	SC00029	An agent named Initiator, takes the role of manager to have a set of tasks performed by single or multiple agents known as the Participants. For certain tasks, any number of the Participants may respond with a proposal to the Initiator. Negotiation then continues with the agents that proposed to the Initiator.
Iterated Contract-Net	SC00030	It is an extension of the basic Contract-Net,but it differs by allowing multi-round iterative bidding.
Brokering	SC00033	A broker is an agent that offers a set of communication facilitation services to other agents using some knowledge about the requirements and capabilities of those agents. A typical example of brokering is one in which an agent can request a broker to find one or more agents who can answer a query.
Recruiting	SC00034	A recruiter is a form of broker, which offers a set of communication services to other agents by using knowledge about the requirements and capabilities of those agents.
Subscribe	SC00035	The Initiator is an agent that starts the interaction with a subscribe message with reference of the objects of interest.The Participant agents process the subscribe message and if they accept or refuse the query request.
Propose	SC00036	The Initiator agent sends a propose message to the Participant agents indicating that it will perform some action if the Participant agrees.Participants can responds by either accepting or rejecting the proposal, communicating this with the accept-proposal or reject-proposal communicative act.

4.1.2. MESSAGE TRANSPORT PROTOCOL IMPLEMENTATION

The FIPA-MTP specifications consider general aspects of communication protocol's implementation, for instance, interface definition, message envelope syntax, and addressing scheme. For specific details of that implementation, every protocol shall provide a set of functionalities that satisfy the needs of the application and the agent platform requirements. Any message transport protocol may use an alternative internal representation to describe a message envelope, but it must express the same terms, and represent the same semantics of the Agent Interaction Protocols (AIP) in Table 4.2.

For that reason, the FIPA-ACL messages require lower-layer protocols for supporting these AIP. FIPA does not restrict the use of any protocol for MTS, but it documents specification for three commonly used Message Transport Protocols: IIOP, WAP and HTTP. However, additional MTPs have been implemented in MAS-based software design. For example, XMPP with SPADE and JAVA-RMI with JADE.

Most of the MAS platforms reported in the literature are intended for mobile applications and distributed robotic systems. According to the author's knowledge, this dissertation is the first one to propose adopting an agent-based approach for implementing the onboard software of satellites. That requires establishing a reliable message transport protocol that can be implemented with space-qualified technology that fits the need of an agent platform. One of these emerging distributed communication protocols is CANopen, which offers the performance and reliability required by satellites systems. CANopen is a high-level communication protocol and a device profile specification that is based on the CAN protocol.

Using the Open Systems Interconnection (OSI) model to describe the proposed communication stack, CAN protocol covers the physical layer and the data link layer, while CANopen implements the network, transport, session, presentation and application layers. The physical layer specifies the lines used as well as the voltage levels. The data link layer is used to encapsulate the messages into frames. CANopen describes the implementation of the top five layers. For the network layer, it implements the addressing and routing schemes. For transport, it delivers end-to-end reliability. The session layer provides synchronization for the transmission of messages, while the presentation includes data codification at a high level for being used at the application level. Finally, the application layer describes how to configure, transfer and synchronize the CANopen devices.

This Chapter proposes the implementation of a CAN-based MTP for supporting MAS-based onboard software on satellite systems. For that purpose, it suggests to use CANopen for the implementation of the data signaling and data transport protocol. The proposed MTP was based on the implementation of the Process Data Object (PDO) protocol on top of CANopen. More details about the motivation of the selection of CANopen for satellite systems can be found in the work of Orsel (2016).

There are two main advantages of adopting the PDO protocol as the MTP in multi-agent systems for satellites. Firstly, its bus utilization efficiency is not affected by the ACL message structure, and the data bus performance can be controlled by the bus operation parameters, as demonstrated in the work of Orsel (2016). Secondly, its implementation flexibility, since it enables three different communication design patterns for producer-consumer communication architectures. These are event-driven, remotely requested and synchronous/asynchronous transmission. This capability makes PDOs suitable to the needs of space systems, particularly for the AOCS subsystem.

On the other hand, there is a drawback related to the maximum data length that can be transmitted at the time by a PDO. This size is up to 8 bytes per PDO, which requires having multiples PDOs to represent a single the ACL message. That can demand extra work to the ACC in the agent platforms to split and assemble messages for their transmission. The following section describes in detail the proposed communication architecture.

4.2. SOFTWARE COMMUNICATION ARCHITECTURE

The goal of proposing a communication architecture for MAS-based software is to describe the interface in which ACL message objects are converted into low-layer protocol objects that are sent and received over a distributed data bus that connects subsystems and components within the spacecraft bus.

The selected architecture requires addressing the following top-level requirements:

1. The selected protocol shall be message-oriented to be compatible with the object-oriented approach required by MAS.
2. The selected protocol shall minimize the communication overhead between ACL messages and low-level message objects.
3. The selected protocol shall provide built-in features for FDIR.
4. The selected protocol shall provide a high Technology Readiness Level (TRL) for space systems.
5. The selected protocol shall provide synchronous operations capabilities to ensure determinism of critical subsystems within the spacecraft.

For low-level implementation only CAN can satisfy these requirements. Therefore it was selected for physical and data link layers as mentioned above. The process to select a higher layer protocol and a design configuration that satisfy these requirements was conducted and documented by Orsel (2016) in his Chapter 4. From that study, the combination of CAN and CANopen outperformed CAN flexible data rate and CAN aerospace, so that that combination was selected to propose a communication architecture for MAS-based software onboard a satellite. Figure 4.2 synthesizes the communication architecture proposed for implementing MAS-based software in distributed computing systems onboard satellites. It assumes that each subsystem or component connected to the distributed communication bus can execute an agent platform which logically contains a MAS-based software application that interacts with other subsystems and components using FIPA-ACL messages.

For communication within the same agent platform, agents rely on the IPMT service implemented by the agent platform, whereas for inter-platform communication the ACC is in charge of taking the ACL messages and mapping them into a set of PDO messages objects. These messages are linked together into a producer-consumer software design pattern supported by the CANopen implementation.

Some applications require more than 512 PDO objects to implement their communication interface. According to Lawrenz (1997), CANopen provides a PDO that can be used as a multiplexed (MPDO) to address that concern. However, its use is not recommended since it causes overhead on the protocol, therefore, reducing bus performance.

The implementation of the proposed architecture also requires providing the operating system onboard with a software device driver that satisfies the CANopen specifications. For instance, CANopenNode provides a free and open source of CANopen stack. It runs on different micro-controllers, as a standalone application or with Real-Time Operating System (RTOS).

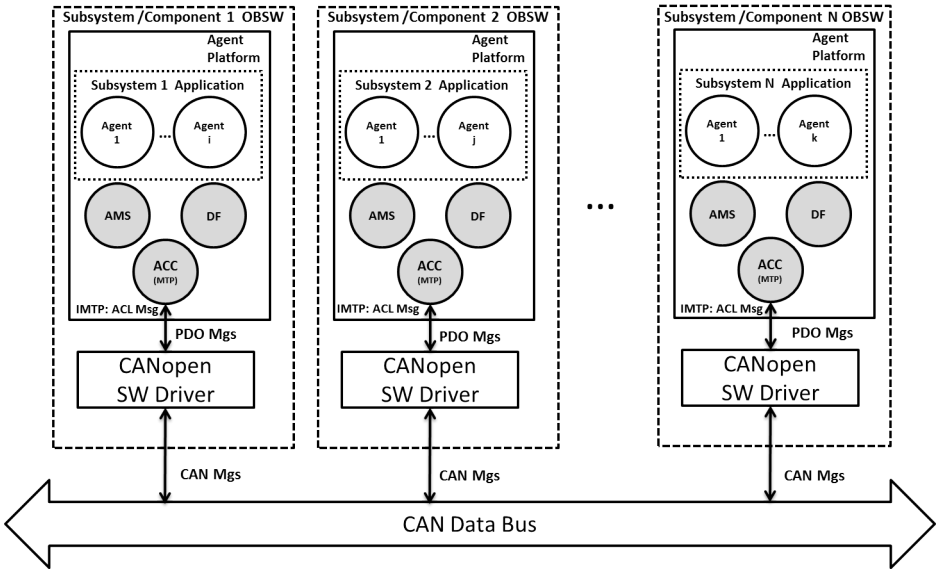


Figure 4.2: Communication architecture proposed for MAS-based software applications used on spacecraft.

The configuration of CANopen is described as follows. Firstly, the device model is explained. Figure 4.3 depicts the communication device model that is proposed for the implementation of distributed communication in MAS-based software for satellites. It consists of three main blocks, which include the communication interface, the object dictionary, and the application process. The communication interface provides the ACC with a set of objects that enable the implementation of several communication design patterns on top of the CAN bus. For instance Service Data Objects (SDO) are commonly used in server-client configurations, while PDO are intended for producer-consumer scenarios. Additional objects are provided by this interface for synchronization purposes as well as for network management and fault detection.

The object dictionary enables the logical addressing scheme for mapping applications process objects, in this case, ACL messages to interface objects such as PDO. It describes all data types, communication objects and application process objects used by the agent platform to communicate with external parties. The application process block enables the device functions within the agent's platform. These are encapsulated in the ACC agent as a cyclic behavior. The ACC provides an interface with the platform's IPMT for enabling communication among agents regardless of the agent platform where they are deployed.

Then, the emergency and synchronization objects are implemented. These are used to convey emergency messages related to the functioning of each node, and providing a time reference on the bus, respectively. Finally the network management object is configured to control of the communication state of network nodes and node monitoring. Once these objects are configured, PDOs are used to transfer real-time data, so the network is fully operational.

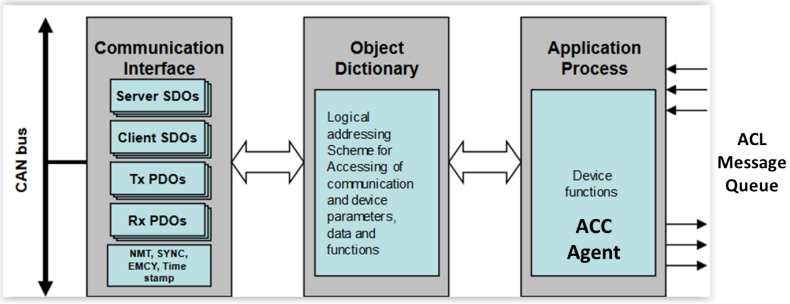


Figure 4.3: CANopen device model adapted from Tortosa López and Roos (2005) for its implementation with an agent-based approach.

4.3. AOCS CASE STUDY

This section is intended to show the feasibility of the architecture depicted in Figure 4.2 using a case study with the AOCS of a satellite. It assumes a producer-consumer communication pattern with multiple components sending information over a distributed communication bus and a single agent platform running on the AOCS onboard computer. The case study also assumes the implementation of a synchronous communication bus with a controllable communication period that varies from mission to mission. It simplifies the mapping of ACL messages to maximize the use of PDO objects and to keep the simulation implementation more straightforward.

4.3.1. AOCS REFERENCE ARCHITECTURE

The simulation model for the satellite aims to emulate the AOCS reference architecture shown in Figure 4.4. The AOCS reference architecture intends to implement the communication architecture described in Section 4.2. In the diagram of Figure 4.4, the AOCS onboard computer is in charge of providing the processing capabilities for attitude and orbit control using a multi-agent systems-based application running on the AOCS.

In Figure 4.4, the communication between components is divided into two categories: peer-to-peer (dotted lines) and distributed (solid lines) communication. Traditionally, peer-to-peer communication is required to interface highly sophisticated sensors and actuators, for instance, some GPS receivers use an RS-422 interface to communicate with the navigation computer as shown by Gill et al. (2001). Distributed data buses are used for less complex and more abundant devices as described by Arruego et al. (2009). From Figure 4.4, its clear that most of the AOCS sensors are connected to the estimation agent via the spacecraft communication bus.

Small satellites, in particular, CubeSats, are more constrained regarding volume and power. That is a motivation for implementing common interfaces for internal spacecraft communication, usually in the form of linear bus topologies. For example, Bouwmeester et al. (2010) used a distributed data bus to simplify the physical interface between multiple components and subsystems for the Delfi-Next satellite. Several other satellites take advantage of distributed communication, but this work focuses on those with volume and power constraints.

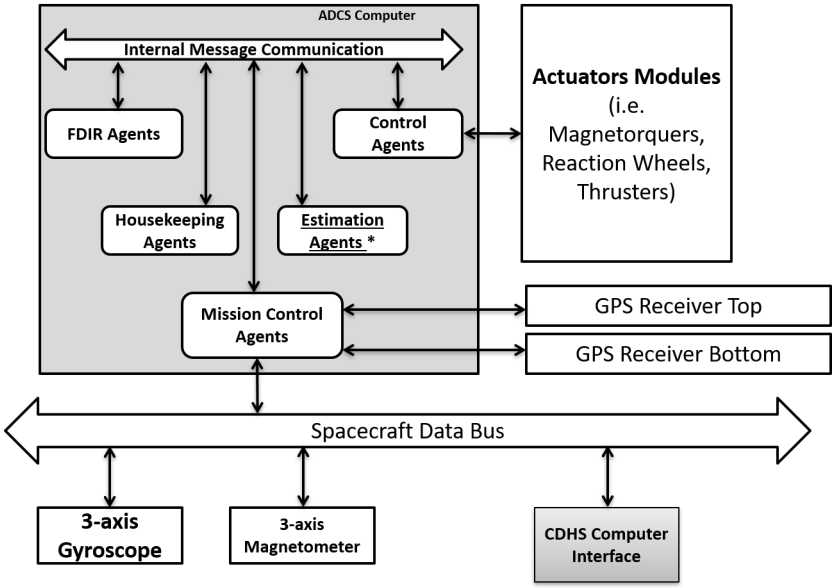


Figure 4.4: AOCS reference architecture for an agent-based software implementation.

A simulation model using $MATLAB^{TM}$ Simulink tools was necessary to implement the data delays in the spacecraft communication bus accurately. For that purpose, the CAN protocol was selected to match the communication device model from Figure 4.3. CAN protocol also is one of the most promising fault-tolerant distributed communication protocols being adopted in the micro/nano-satellite community as shown in the work of Kimm and Jarrell (2014). One of the main advantages of CAN protocol is its maturity level in harsh environments, such as automotive application as discussed by Khurram and Zaidi (2005). The main drawback of CAN is its asynchronous operation that is compensated by the higher layer protocol implementation with CANopen. Also CAN does not use a master/slave communication pattern. Instead it uses medium access control is distributed among the nodes in the network as discussed by Plummer et al. (2003).

Different implementations of CAN, for instance, Flexible Time-Triggered CAN combines Event-and Time-Triggered capabilities to enable flexible operation in industrial systems, which can also be adapted for space applications. The following subsections elaborate on the implementation of a simulation model for synchronous CAN protocol, as well as the implementation of the sensor model and the traffic generation model used to simulate the satellite's high throughput operation scenarios.

4.3.2. AOCS MEASUREMENT MODEL

Most of the estimation algorithms for satellites work over a linearized system model so that they can be implemented on the AOCS computer. The use of an Extended Kalman Filter (EKF) is proposed to predict and update the spacecraft state from a non-linear

model as described in detail in Appendix C.

State estimators using distributed communication architectures are known to have fading channels that impact their performance. The measurement model is then extended to accommodate this effect as presented and discussed by You et al. (2015) and shown as follows

$$\mathbf{z}(k) = \boldsymbol{\xi}_k \mathbf{y}(k) + \boldsymbol{n}_k \tag{4.1}$$

where, $\mathbf{n}_k \in \mathbb{R}^L$ is also an additive white noise accounting for delays in the measurements, and $\xi_k \in \mathbb{R}^{LxL}$ is a diagonal matrix accounting for the fading effects on the i_{th} link of the communication bus.

This structure is introduced as

$$\begin{aligned} \boldsymbol{\xi}_k &= \text{diag}\{\xi_{1,k}, \xi_{2,k}, ..., \xi_{i,k}\} \\ \xi_{i,k} &= \gamma_{i,k}\Omega_{i,k} \end{aligned} \tag{4.2}$$

where, $\xi_{i,k}$ represents the faults mechanisms for the communications channel. In this expression $\gamma_{i,k}$ is a Bernoulli process that models the arrival of measurements, and $\Omega_{i,k}$ describes the signal fluctuation due to channel's performance degradation as discussed by Sinopoli et al. (2004).

The arrival of measurements is considered as a success if the data arrives before a delay threshold, or as a failure if it exceeds the threshold or if it gets lost during its propagation. The probability that describes the arrival of measurements is shown in the following expression

$$\begin{aligned} Pr\{\gamma_{i,k} = 1\} &= \alpha_i, \\ Pr\{\gamma_{i,k} = 0\} &= (1 - \alpha_i) \end{aligned} \tag{4.3}$$

where, according to Marshall and Olkin (1985), the parameter α_i determines the probability of an arrival to be successful.

This delay is not accounted within the sensor model, but it is modeled as a bus effect, as depicted in Figure 4.5. The delay of measurements is assumed to produce an additive error that can be statistically estimated as

$$\mathbf{n}_k = \begin{cases} 0 & \text{Nominal Region,} \\ Norm(\mu_S, R_S) & \text{Saturated Region} \end{cases} \tag{4.4}$$

Under nominal conditions, the effect of the delay errors is expected to be negligible, whereas when the communication channel reaches a saturation point, the delay is modeled as a normal distribution centered to a mean value for each sensor μ_S with a measurement variance of R_S. The representation of the measurement delay error as a random variable makes it possible to add it to the measurement model of the Kalman Filter used in the attitude estimation model introduced in Appendix C.

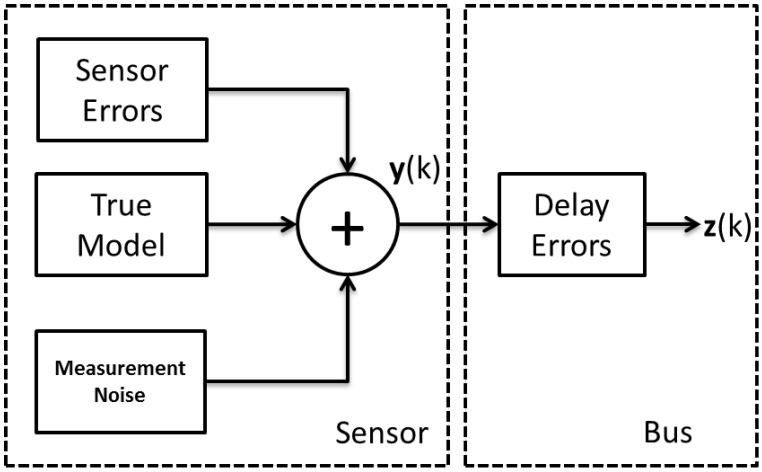


Figure 4.5: Block diagram for simulation model of sensors connected to the communication bus.

Using the above's definition, the AOCS estimation model can be re-written to include the effects of faulty communication channels and measurement delay as in the work of Zhang and Jiang (2003) using the following expression

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k)] + \mathbf{w}(k) \\ \mathbf{z}(k) &= \boldsymbol{\xi}_k \mathbf{h}[\mathbf{x}(k)] + \boldsymbol{n}_k + \boldsymbol{\nu}_k \end{aligned} \tag{4.5}$$

4.3.3. TRAFFIC INJECTION MODEL

It was necessary to generate additional traffic to emulate different satellite operation loads and characterizing the performance of the communication bus. The traffic injection function produces a burst which follows a Poisson Process with a Inter-Arrival Time (IAT) that is a function of an average message arrival rate λ in messages per second described as

$$IAT = \begin{cases} \infty & \text{Nominal,} \\ \frac{1}{\lambda} & \text{Additional Injected Traffic} \end{cases} \tag{4.6}$$

This traffic is added to the nominal traffic in the communication bus that is exchanged between satellite sensors and subsystems. When the IAT is ∞, there is no additional traffic injected in the communication bus.

4.3.4. COMMUNICATION BUS LOAD MODELING

Saturation refers to the capacity of a communication channel to deal with incoming messages beyond its nominal capacity. It can be measured by monitoring the Bus Utilization (BU), which is directly related to the number of incoming messages and the implementation parameters of the communication channel, as well as, the network size.

The bus utilization is defined as the relation between the time required to transmit a group of messages divided by the total time available in the bus to complete this task in every transmission cycle. For the CAN protocol an estimated value for the worst case transmission time is described by Broster and Burns (2001).

Bus utilization analysis requires accounting the total number of messages transmitted over the communication channel per synchronization event. Assuming there are N nodes sharing messages with length M_L in bits over the communication bus, the message volume is defined by the number of messages and their length in bits as

$$M_S = \sum_{i=1}^{N} M_{Li} \tag{4.7}$$

where, M_S is the total number of bits that are sent over the communication bus per transmission cycle. It is assumed that the communication bus has a constant synchronization period T_{Sync} in seconds and a constant bus data rate D_R in bits per second.

This assumption allows quantifying the impact of the topology configuration in the BU as

$$BU = \frac{M_S}{T_{Sync} D_R} + \frac{M_L}{D_R \lambda}. \tag{4.8}$$

From (4.8), the link between bus utilization and the number of sent messages is directly proportional. Also, note that the increase in bus utilization is dependent on the average message arrival rate λ used to describe the additional traffic injected.

It is necessary to keep in mind that (4.8) assumes the channel is operated under nominal conditions (not saturated), which means that there is enough time to transmit the total number of messages between synchronization events. That is critical to avoid losing messages with lower priority.

4.4. CASE STUDY IMPLEMENTATION

This section focuses on the implementation aspects of the simulation model developed to characterize and quantify the effect of propagation delays on sensor's measurements received at the AOCS computer connected to the distributed communication bus in Figure 4.4.

4.4.1. CAN CHANNEL IMPLEMENTATION

The implementation of the communication channel was divided into two parts: one to describe the mechanisms to access the physical channel known as Medium Access Control (MAC), and the second one to implement the channel controller and its interface with the application layer.

MEDIUM ACCESS CONTROL

Since CAN protocol operates using a bus topology, it is necessary to establish a method for all the nodes to access the communication medium. CAN establishes that the physical layer is specified through the standard ISO 11898-3. The specification includes data rates up to 1000 kbps. In the simulation model, the channel was implemented using the

discrete time simulation approach to make it synchronous. This implementation consisted of two queues: one for receiving and ordering messages by priority (arbitration) using the internal message ID given by its node address, and the other to queue and broadcast the messages to the nodes connected to the bus.

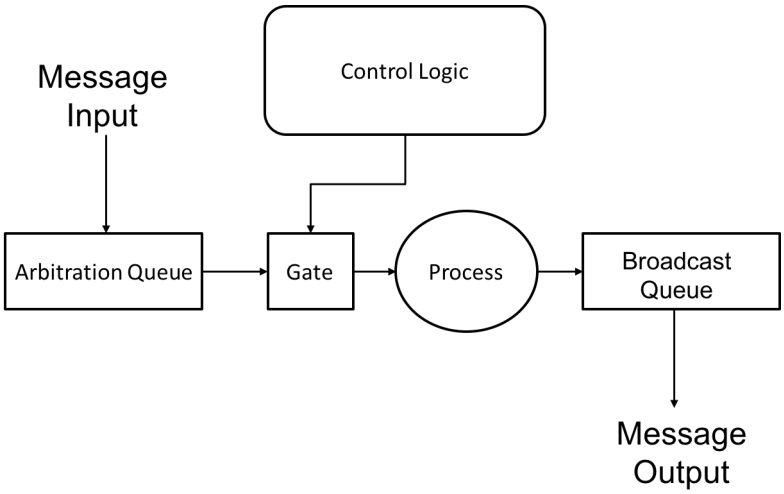


Figure 4.6: Block diagram for the CAN channel implemented in the simulation model.

In Figure 4.6, the block diagram for CAN channel implementation is shown. In addition to the arbitration and broadcast queues, there is a control logic block in charge of executing the medium access control algorithm to determine when a message is allowed to be processed. The process block simulates the time required by the message entity to propagate over the physical channel. This parameter is called process time P_T, and it is fixed as a function of the channel data rate capacity D_R in bits per second, the number of bits per CAN message NB_{CF} and the channel physics NB_P (e.g. cable length) as

$$P_T = \frac{NB_{CF} + NB_P}{D_R} \tag{4.9}$$

The simulation model operates using two data rates. One called baseline at the speed of 500 kbps and the high speed at 1000 kbps. The size of the CAN payload data was defined a constant of 64 bits for a total encoding length of 113 bits per CAN message. This size includes penalties for channel physics, and it excludes stuffing bits. The control logic block also included the possibility of establishing a percentage of messages lost in the channel as a fixed parameter to model fading channel faults.

CAN CONTROLLER

On top of the physical model abstraction, a CAN controller model was synthesized for packing and unpacking data transmitted over the bus. It consisted of a transmitter and a receiver. The transmitter took the incoming data from the upper application layer and

framed it into a CAN message following the standard CAN framing structure. Additionally, each message contained a time stamp that was used to calculate the delay of a message on arrival. Also, the transmitter was provided with a queue to hold messages when the communication channel was busy. The length of that queue was set to a capacity of 3 messages, to make it consistent with the size of the buffer of a commercial-off-the-shelf microcontroller (SMT32F405) used as a reference. The transmitter also was in charge of generating the synchronization for the transmission of the messages over the bus. It was controlled during the simulation with the channel synchronization period T_{SYNC} parameter. Figure 4.7 summarizes the CAN controller transmitter as a block diagram. The scheduling of transmission and reception was assumed ideal as in the work of Tindell et al. (1994). The message output of this diagram feeds the broadcast queue described in Figure 4.6.

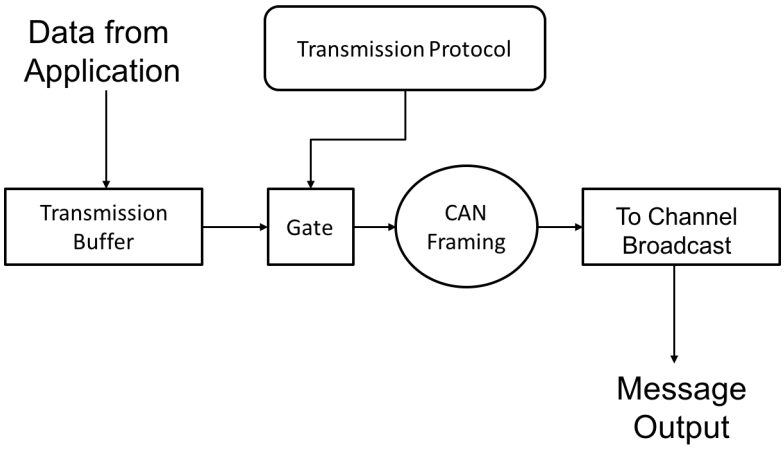


Figure 4.7: Block diagram for implementing the transmitting model of the CAN controller.

The receiver consisted of a reception buffer with a capacity for three incoming messages from the communication channel layer. Later, a reception protocol was applied to these messages to filter them by node ID and verifying its data integrity using the CAN cyclic redundancy check specification. Then, the messages were decoded, and the payload data was retrieved and forwarded to the upper layers at the application level. Figure 4.8 shows the block diagram for the implemented receiving structure. The messages from the channel in this Figure are taken by priority from the arbitration queue in the Figure 4.6.

At this point, the delay Δt of a specific CAN message propagated from its source node at the sensor to the destination at the AOCS computer can be calculated as

$$\Delta t = t_S - t_{AOCS} \tag{4.10}$$

where, the t_S is the time at which the data was packed into a CAN message at the source node and the t_{AOCS} is the time in which the message is unpacked at the destination node in the navigation computer. It assumes that all the clocks in the components are synchronized so that the delay calculation is precise.

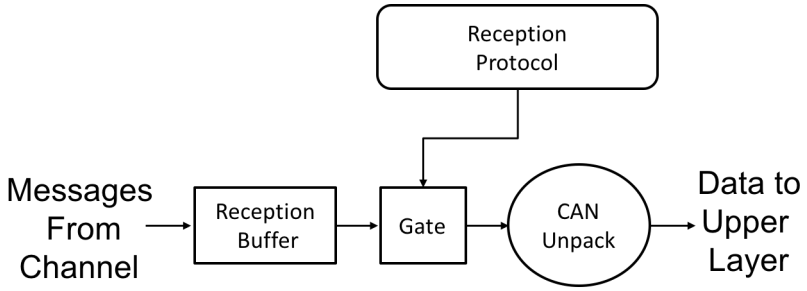


Figure 4.8: Block diagram for implementing the receiving model of the CAN controller.

The CAN controller model required to be provided with the following parameters for its operation: CAN controller ID, payload data size, IDs of subscribed nodes and a range of valid node IDs. This information is used during the run-time to filter the messages each node receive, but also to prioritize the messages received in the arbitration queue.

4.4.2. SENSOR MODEL IMPLEMENTATION

One of the primary objectives of this work is to understand the effect of networked sensor communication in the variability of the measurement as perceived in the estimation algorithm. For that purpose, a model for the measurements was introduced in (4.5) for all the attitude sensors connected to the distributed communication bus as depicted in the AOCS Reference Architecture.

Figure 4.5 shows the block diagram for implementation of each sensor used in the simulation model. In that diagram, the true model propagator is in charge of generating the true value for sun sensors, magnetometers and gyroscope using the spacecraft dynamics and kinematics equations. These measurements include sensor errors such as bias, drift and noise. The measurements are then supplied to the CAN controller for its propagation through the communication bus, where delays can cause additive errors.

4.5. SIMULATION EXPERIMENTS

This Section is divided into two parts. The first Subsection illustrates the use of the proposed communication architecture from Figure 4.2 to describe satellite operation scenarios with small satellite missions. The second subsection elaborates on the experimental design and configuration used to collect data on the performance of the communication bus. The simulation experiments were carried out using $MATLAB^{TM}$ Simulink 2016.

4.5.1. SATELLITE OPERATIONS SCENARIOS

Two operation scenarios are considered for this case study. The first one involves an increase in the communication activities over the communication bus due to telemetry download, and the second relates to traffic increase in the bus due to additional data generated by ADCS sensors onboard of satellites with optical downlink communication. The primary objective of this Subsection is to propose practical cases to verify and vali-

date the bus utilization model described in Section 4.3.4, and demonstrating the feasibility of the communication architecture presented in Section 4.2 for MAS-based software on satellites.

TELEMETRY DOWNLOAD

The Delfi-Next satellite was launched in 2013. It was the second in a series of small satellite projects by Delft University of Technology (TU Delft) in the Netherlands. Delfi-Next was a triple unit CubeSat with a size of 10 cm x 10 cm x 34 cm. It had an active attitude determination and control subsystem and a high-speed S-band transmitter communicating over a distributed linear bus with the rest of the spacecraft subsystems as described by Guo et al. (2016).

The Delfi-Next satellite collected more than 300 telemetry parameters every two seconds. These parameters were broadcasted to the amateur radio network on the ground where they were collected and stored into a database for further analysis. Also, the telemetry packets were stored locally in a database implemented on the onboard computer. The database was synthesized into a telemetry file that was intended to be downloaded to the ground station in Delft by using the S-band transceiver onboard.

This scenario is inspired by Delfi-Next operational conditions and architecture. It consists in simulating the telemetry file download during one pass of the satellite over the ground station at least one time per day. The Delfi-Next telemetry file contained information from 316 satellite parameters that were mapped into 50 PDO/CAN messages for its transmission from the OBC to the S-band transceiver over the distributed data bus.

The telemetry file is expected to be downloaded in 20 seconds while the satellite passed over the ground station in Delft. Assuming each CAN message transport 8 bytes, the total size of the telemetry file is about 8 KB. If the telemetry file is not downloaded entirely, the missing part will be added to the file for the next satellite which will increase its size.

The data rate of the communication bus in the satellite required to be configured at 1000 kbps to satisfy the download time. The arrival rate of CAN messages at the S-band transceiver was modeled to vary from 50 CAN messages per second in the best case scenario to 20000 CAN messages per second as a worst case. That is done to describe a wide range of telemetry file sizes up to 8 MB.

The traffic generated due to the transmission of the telemetry file from the OBC to the S-band transceiver is added to the nominal traffic using the data bus of the satellite. That traffic is considered additional injected traffic during the simulation of the case study. That creates different utilization levels in the data bus of this spacecraft which are characterized and shown later. The simulation experiments consider the operation of the data bus for additional injected traffic that covers the whole range of telemetry file sizes described above.

ADCS TRACKING FOR OPTICAL COMMUNICATION

Optical communication payloads require a high satellite pointing accuracy. For these satellites, the ADCS requires the implementation of both coarse and fine pointing algorithms to support the optical communication performance. In the work of Nguyen et al. (2015), the ADCS subsystem was enhanced by adding an optical beacon detector

to provide on-line tracking adjustment and calibration to the optical transmitter with fast-steering mirrors.

This simulation scenario assumes modifying the ADCS reference architecture proposed in Figure 4.4 to include the optical beacon detector and the fast steering mirrors connected to the distributed data communication bus. During the simulation the traffic generated by the beacon detector is injected to the communication bus to assess its impact on the estimation algorithm onboard the ADCS computer. The data rate of the communication bus is assumed fixed at 500 kbps.

Figure 4.9 shows the architecture for the enhanced ADCS configuration. For the coarse pointing, the Extended Kalman Filter fuses data from the ADCS sensors at two Hz, while for the fine pointing mode, the beacon detector and the centroid algorithm are working at 10 Hz using the same bus to communicate with all the other subsystems and components. The additional injected traffic was defined to vary in the range of 50 to 10000 CAN messages per second to assess the impact of additional traffic in the CAN bus utilization.

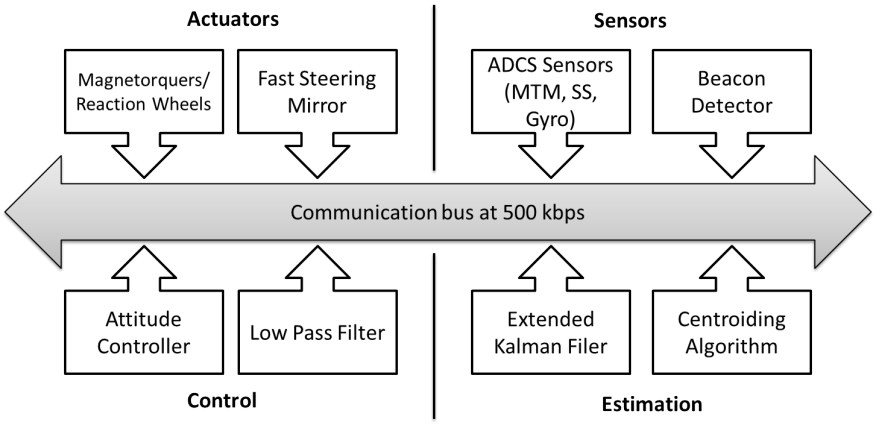


Figure 4.9: Setup for the ADCS tracking scenario with additional hardware and software components.

4.5.2. SIMULATION CONFIGURATION

The spacecraft implementation at simulation level consisted of a CAN network with up to 16 nodes including a 9-axis integrated IMU, sun sensors, thermometers, magnetometers. Also included three reaction wheels, an optical beacon detector, an ADCS computer, a CDHS computer, and a downlink/uplink communication module. The measurement error was determined indirectly by monitoring the change in the sensor's measurement variability. The hypothesis is that delayed measurements lead to changes in the value of the normalized variance of the sensors transmitting over the bus.

The normalization is calculated taking the measurement received the ADCS computer and dividing it by the measurement variability at the sensor, so that the observed changes are related to saturation in the communication bus. It assumes that the variance of the measurements at the sensor source shall keep constant over time.

The additionally injected traffic is used to represent the dynamic behavior of the communication bus during telemetry download and ADCS tracking described in Section 4.5.1. The data rate D_R represents internal operation modes (baseline and high-speed) for the communication bus that can be controlled during satellite operations.

The simulation setup was chosen based on two conditions. Firstly, for data rate and synchronization period, it was based on technology constraints for commercial CAN transceivers and controllers (e.g, SMT32F405 microcontroller). Secondly, the sensor sampling period and the network size were based on previous ADCS configurations for CubeSats, for instance in the work of Vinther et al. (2011).

The inputs for the simulation experiments are summarized in the Table 4.3.

Table 4.3: Description of the inputs considered during the implementation of the data bus simulation

Name	Description	Input Type	Values Used
NUM_NODES	Number of nodes connected to the communication bus	Parameter	16
M_L	CAN message length in bits per message	Parameter	113
TX_QUEUE_SIZE	Number of CAN messages that can be stored by the transceiver for their transmission over the communication bus.	Parameter	3
RX_QUEUE_SIZE	Number of CAN messages that can be stored by the transceiver for after their reception in the communication bus.	Parameter	3
T_{SS}	Period used to sample the sensors connected to the communication bus.	Variable	0.1 s and 0.5 s
T_{Sync}	Period used to synchronize the operation of the communication bus.	Variable	0.01 s and 0.05 s
D_R	Channel's data rate for the operation of the communication bus.	Variable	500 kbps and 1000 kbps
AVG_INJ_TRF	Additionally, average injected traffic into the communication bus	Variable	50 to 20000 Msg/s and 50 to 10000 Msg/s

The simulation experiment was designed following a 2^k full factorial design. The output variables collected for both satellite operation scenarios were:

- Mean communication bus utilization [%]
- Maximum communication bus utilization [%]
- Mean sensor measurement delay [s]
- Maximum sensor measurement delay [s]
- Measurement variance at source
- Measurement variance at OBC

The bus utilization and sensor measurement delay were used to characterize the impact of injecting additional traffic into the communication bus, whereas the sensor's measurement variance was used as an indicator for determining the impact of delays in the estimation algorithm performance.

4.6. SIMULATION RESULTS AND ANALYSIS

This section is divided into five parts to present and discuss the findings of the simulations carried out. It also proposes a flow diagram to mitigate saturation effects on the communication bus by adopting a dynamic configuration of channel's parameters during the run-time.

4.6.1. BUS UTILIZATION

The Bus Utilization (BU) was obtained as a function of the additionally injected traffic that depends on λ, the data rate D_R, and the bus synchronization period T_{Sync} for both operation scenarios. The goal of these simulations was reproducing different bus load conditions to characterize BU under different bus saturation levels. The saturation levels represent different operational conditions described in Section 4.5.1. In Figure 4.10 and Figure 4.11, the utilization for the CAN bus in the ADCS tracking case at 500 kbps, and the telemetry download case at 1000 kbps are shown, respectively. The theoretical curves calculated from (4.8) are included in the bus utilization profiles for comparison purposes. Both bus utilization figures were marked with tags to highlight specific behaviors for their different configurations.

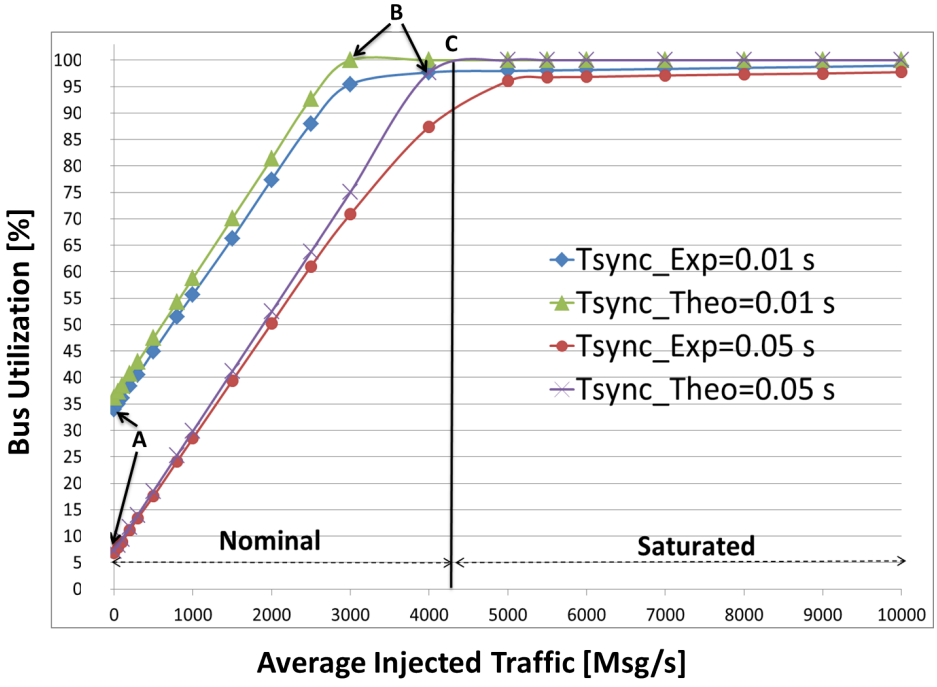


Figure 4.10: Bus utilization for the ADCS tracking operation scenario at 500 kbps with T_{Sync} values of 0.01 s and 0.05 s obtained using both the analytical (Theo) and the simulated (Exp) models for a CAN network consisting of 16 nodes.

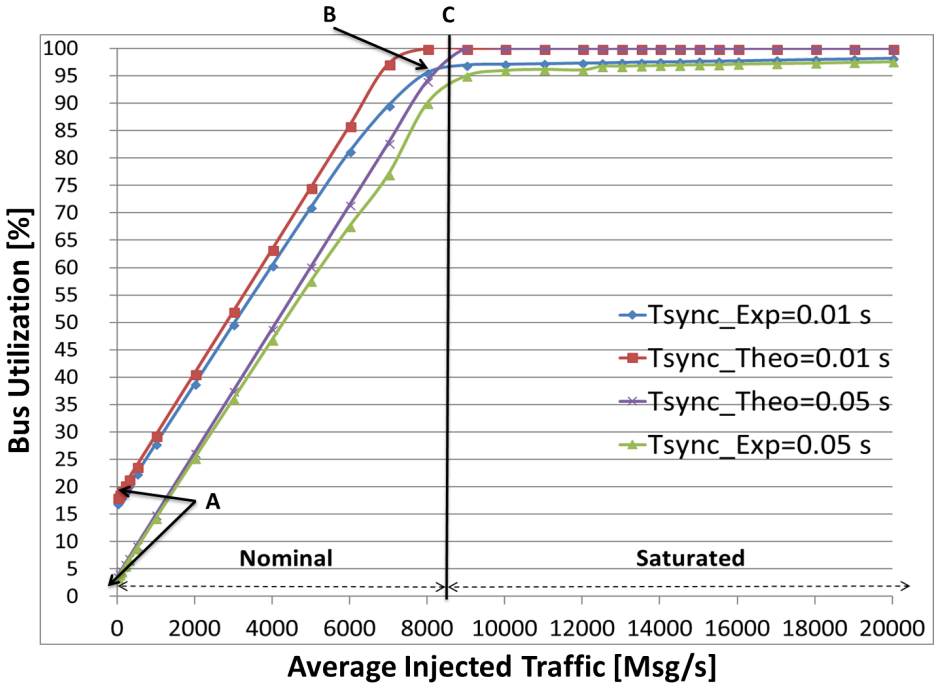


Figure 4.11: Bus utilization for the Telemetry Download scenario at 1000 kbps with T_{Sync} values of 0.01 s and 0.05 s obtained using both the analytical (Theo) and the simulated (Exp) models for a CAN network consisting of 16 nodes.

Firstly, there is the segmentation of the curve into two operation regions namely nominal and saturated. The limit between the nominal and saturated region is defined by the end of the linear behavior in the bus utilization (point B in the profile). The boundary between the nominal and saturated region is marked by the maximum physical capacity of the channel (point C), that can be determined analytically for each T_{Sync} and D_R combination. It is important to note that the experimental values are consistent with the ones calculated with the analytical model from (4.8). It can also be observed in both figures that the BU for T_{Sync}_Theo is always higher than the BU for T_{Sync}_Exp for all the bus configurations. It was found that the average of the absolute difference was less or equal than 2.5% for all operation scenarios, and it increased as the communication bus got saturated.

The maximum physical capacity was calculated dividing the channel data rate by the number of bits per CAN message. In the simulation model, each CAN message contained 113 bits, and it was kept as a constant parameter along the case studies. Therefore, the maximum physical capacity of 500 kbps and 1000 kbps were determined to be 4100 and 8200 messages per second at T_{Sync} = 0.05 s, respectively. Beyond this physical limit, the bus controller only allows the higher priority messages to be transmitted, while the lower priority components are held to access the bus causing an extra delay.

The bus saturation points are influenced by both data rate and channel synchronization period, as described by (4.8). Increasing the T_{Sync} reduces the BU so that more significant the amount of additional injected traffic can be handled by the communication channel before reaching saturation conditions mentioned above. From an implementation perspective, the values for data rate are determined by the technology used in the implementation of the CAN controller and transceiver, while the range of synchronization period can be established depending on the network size (N) and the sampling period for the sensors (T_{SS}) in the spacecraft. For practical cases, it is recommended to use a $T_{SS} \geq 10 T_{Sync}$ to avoid aliasing effects of the communication channel in the sensor measurements as suggested by Franklin et al. (1994).

The third aspect to look into the bus utilization profiles was the starting point of the curve in the nominal region (Point A) at different synchronization periods T_{Sync}. A sensitivity analysis was conducted to understand the effect of the number of nodes connected the bus and the T_{Sync} parameter on the initial value for the bus utilization (IBU). Figure 4.12 shows how the initial value of the BU varies for different bus configurations. The values used on that analysis were a data rate of 1000 kbps, a sensor sampling rate of 0.5 s and three CAN synchronization periods: 0.01 s, 0.05 s, and 0.1 s.

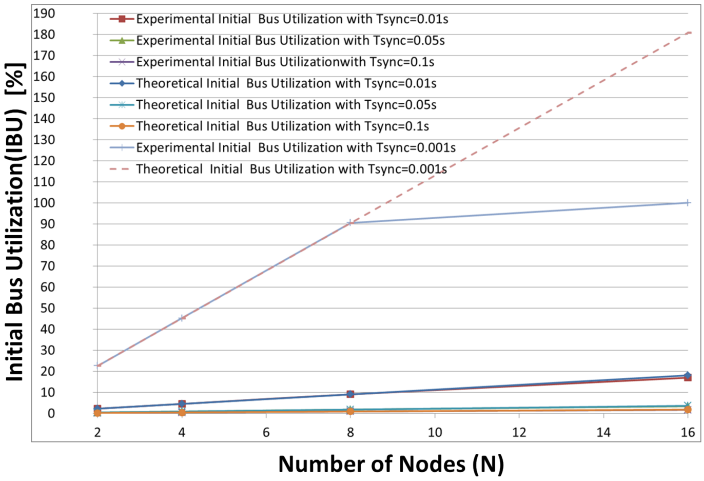


Figure 4.12: Effect of network size in the initial bus utilization for T_{Sync}= 0.01 s, 0.05 s, and 0.1 s.

It is important to note that for all $T_{Sync} \geq 0.01\ s$ the bus utilization scales linearly with the number of nodes, as long as the bus does not reach the maximum capacity. For the sensitivity analysis, no additional traffic is injected. A fixed data rate of 1000 kbps was used so that the bus utilization was a function of the number of nodes in the network and the synchronization period T_{Sync}. It verifies the consistency between the theoretical and the experimental model with an error of less than 1%. It also shows that varying T_{Sync} the slope of the saturation curve can be controlled, which is used later to define a strategy for bus utilization balancing.

4.6.2. MEASUREMENT DELAYS

During the simulations, the mean and the maximum delay for sensors communicating to the ADCS computer over the distributed data bus on the spacecraft were monitored and recorded. It was perfomed for both operation scenarios described above.

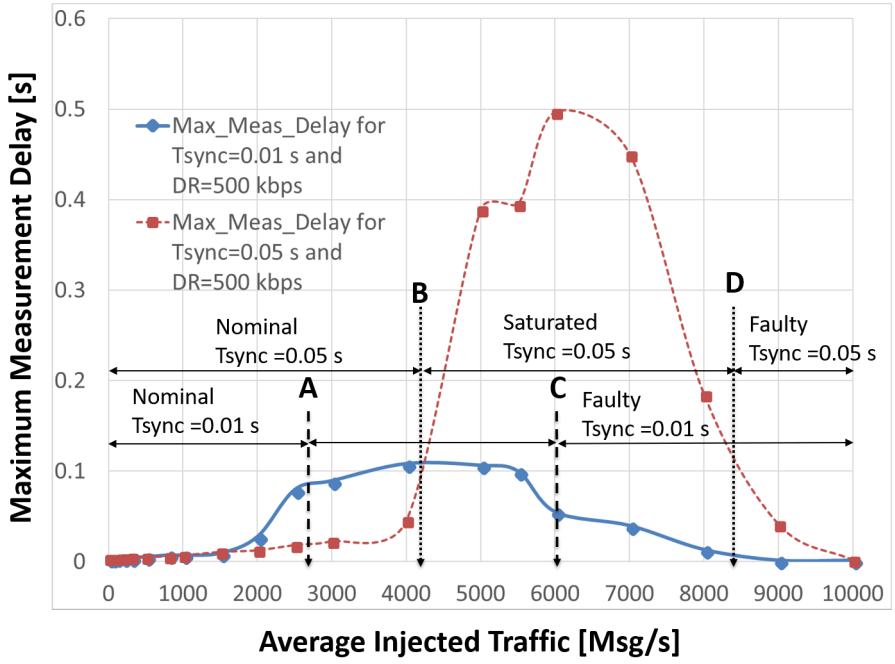


Figure 4.13: Maximum delay observed as a function of the additional traffic injected to the communication bus working at D_R=500 kbps, with T_{Sync}= 0.01 and 0.05 s, respectively.

Figure 4.13 and Figure 4.14 show the effect of channel saturation in the maximum delay observed for both data rates 500 kbps and 1000 kbps, respectively. Each plot shows the delay recorded for T_{Sync}=0.01 s and T_{Sync}=0.05 s as a function of the injected traffic on the communication bus. In both Figures, labels A and B define the saturation point for each T_{Sync} configuration. Label A defines the saturation point for T_{Sync}=0.01 s and label B defines the saturation point for T_{Sync}=0.05 s. It is important to note that for both data rates, when T_{Sync} was increased, also the capacity of the bus to deal higher injected traffic without significantly increasing the delay. This behavior is consistent with the bus utilization model presented in (4.8). However, it was observed that the higher the T_{Sync}, also the bigger the maximum delay observed on the saturated region for both data rates. For design purposes, the maximum T_{Sync} possible is limited by the sensor sampling period T_{SS} requirements as discussed above.

The data rate also affected the maximum delay observed during the experiments. The higher the data rate D_R, the bigger the maximum delay value observed. Labels C and D were used to determine the traffic level under which the simulation model fails. Beyond that point, the delay was observed to fall abruptly to zero because the simulation

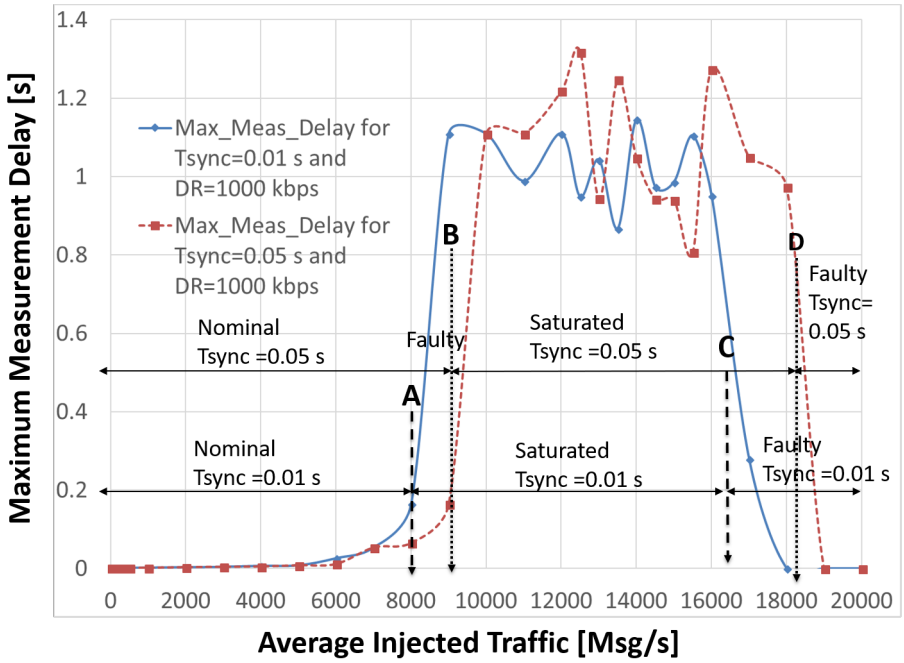


Figure 4.14: Maximum delay observed as a function of the additional traffic injected to the communication bus working at D_R=1000 kbps, with T_{Sync}= 0.01 and 0.05 s, respectively.

in no longer operative. The saturated region in both plots showed that the simulation model can handle approximately the double of the theoretical channel capacity before failing. The oscillating behavior of the delay curves in the saturation region is because the additional traffic uses a random function to generate the exponential inter-arrival times as described in (4.6). Two box-plot graphs were synthesized in Figure 4.15 and Figure 4.16 for both operation scenarios to statistically compare the effect of additional injected traffic and synchronization period T_{Sync} to the measurement delays.

After performing a two-sample t-test for the nominal operation region, there is no statistical significance (difference) for either the data rate and the synchronization period. That means that while the bus operates in the nominal region, the communication bus is not affected by the additional injected traffic.

In contrast for the saturated region, where both mean, and variance of the delay are statistically different. In Figure 4.16, it is clear that there is an effect of saturation in the mean and variance of the delay for the proposed operation scenarios. The effect of the synchronization period is hard to be seen in the box plot, but the numerical test showed a statistical difference based on the P-value reported in during the test.

According to (4.4), the behavior of the measurements delay in the saturated region of the communication channel can be described as a random variable following a normal distribution. That was validated applying the Anderson-Darling Test for normal distribution to the measurement's delay values obtained during the simulation for all the test

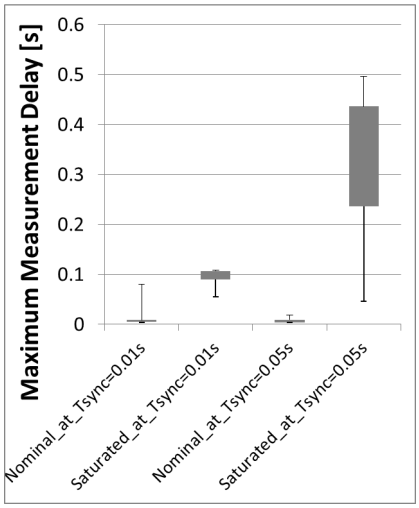


Figure 4.15: Statistical comparison for measurement's delay with the communication bus operating with a D_R = 500 kbps and two different T_{Sync} values.

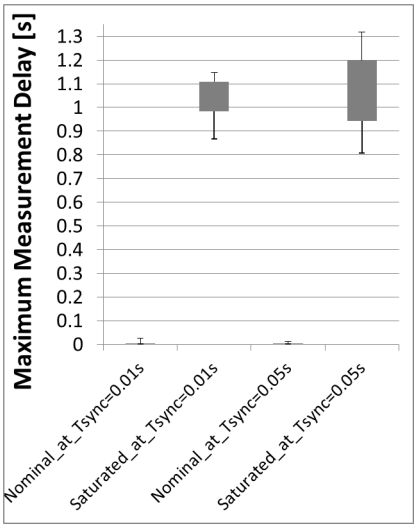


Figure 4.16: Statistical comparison for measurement's delay with the communication bus operating with a D_R = 1000 kbps and two different T_{Sync} values.

cases. It was confirmed that the assumption was valid with a 95% confidence level.

In summary, there is an effect in the delay of the measurement that depends on the operation regime of the channel and the configuration parameters such as data rate and synchronization period. That effect can be described as a random variable that can be included as an additive error in the measurement model of the sensors.

4.6.3. EFFECT OF DELAYS IN MEASUREMENTS VARIANCE

So far, the effects of design parameters such as data rate and synchronization period on the communication channel saturation and the measurement delays within the spacecraft bus have been presented and discussed. From the state estimation perspective, it is necessary to understand the effect of delays in the sensors measurement quality received at the ADCS computer for both operation scenarios. A series of simulation experiments were carried out with the proposed communication bus model to quantify the impact of bus saturation in the variance of the measurements received at the estimation algorithm.

The measurement's variance behavior of the ADCS sensors was monitored and recorded for both operation scenarios in the case study to quantify the effect of delays in their measurements quality. Then, the data collected was post-processed and normalized. The normalization consisted in dividing the measurement's variance calculated at ADCS computer between the variance calculated at the sensor's source, so that if there was not a change due to delay they may be equal at the source and in the ADCS computer.

After that, the change in the normalized measurement variance was plot as a function of the injected traffic. The measurements time stamp was collected both at the source in the sensor, as well as, in the destination in the ADCS computer for both scenarios. Finally, a MATLAB script calculated delays in the data propagation within the communication bus. That script also calculated the variance at the source and at the ADCS computer, so that they can be compared for each sensor connected in the distributed data bus.

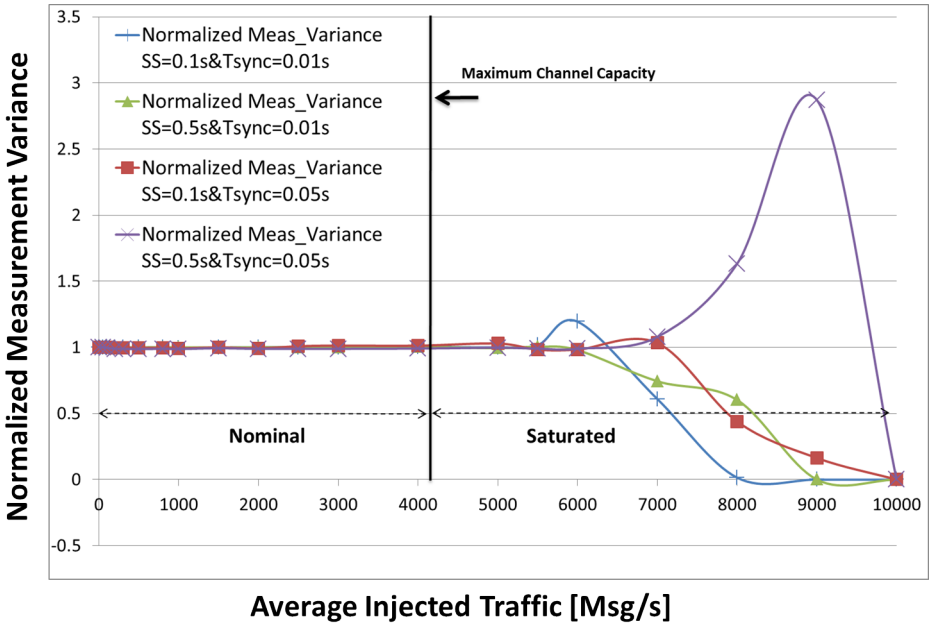


Figure 4.17: Measurement Variability for Communication bus at 500 kbps

Figure 4.17 and Figure 4.18 show normalized measurement variability profiles for

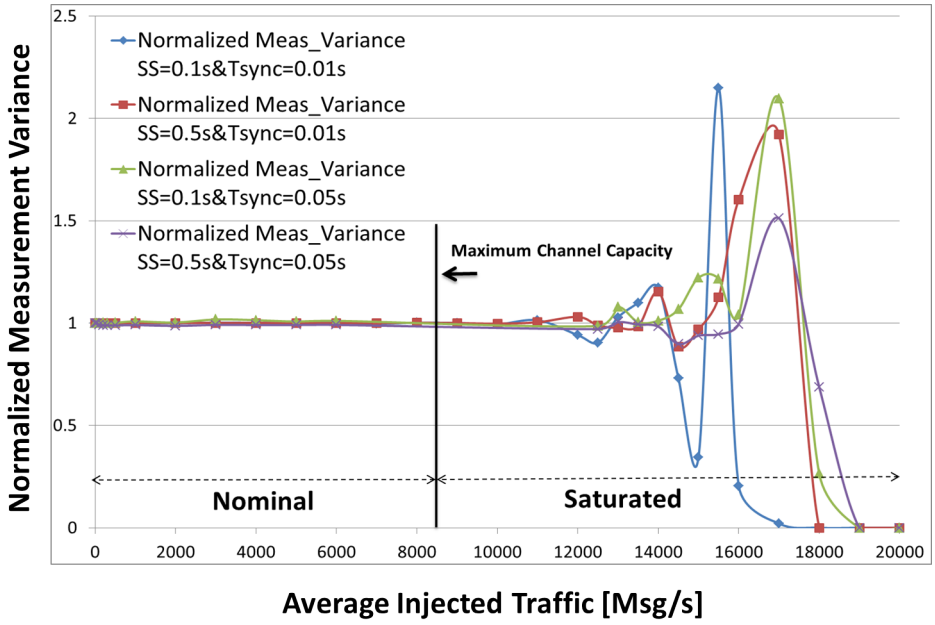


Figure 4.18: Measurement Variability for Communication bus at 1000 kbps

two sensor sampling periods (0.1 s and 0.5 s), two-channel synchronization periods (0.01 s and 0.05 s), and two data rates (500 kbps and 1000 kbps). One can notice that in the nominal region the variability kept constant to the unit value, meaning that in this region the channel configuration does not affect the measurement, and therefore the state estimation performance. This is not the case for the saturated region, where the additionally injected traffic increased the variability of the measurements, and these started to exhibit an oscillating behavior with an absolute difference of average 10%. From his analysis of the variability profiles, it is clear that the estimator is more sensitive to the datarate D_R rather than the synchronization period T_{Sync}.

In Figure 4.18 the oscillation characteristic for 1000 kbps appeared later than for the 500 kbps data rate in Figure 4.17. It is also clear that this variability behavior increased with the amount of injected traffic in the bus, as expected, due to the delay in the arrival of the ADCS measurement samples. Figures 4.19 and 4.20 compare the statistical performance of the communication bus configurations tested during the experiments to study the effect of channel saturation on measurement variability.

From Figure 4.19 and Figure 4.20 there is a noticeable difference in the measurement variability deviation for the saturated region compared to the nominal region. After performing a 2-sample-t-test for both cases there is no significance for the means, but there is for the variance of the measurements. Again, a higher sensibility is linked to the data rate and synchronization period, while there is no effect coming from the sensor sampling period. The additional traffic range was extended until having the simulation model to fail. In the 500 kbps case, the model failed after 10000 mgs/s. For the 1000 kbps,

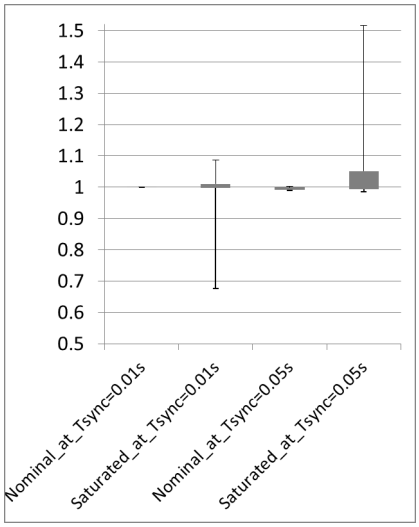


Figure 4.19: Statistical Comparison for Measurement's Variability in Communication bus at 500 kbps

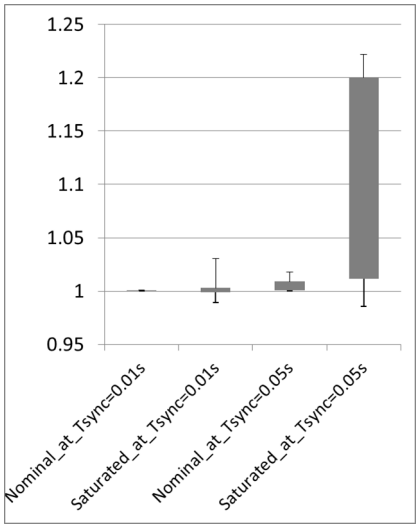


Figure 4.20: Statistical Comparison for Measurement's Variability in Communication bus at 1000 kbps

it failed after 20000 msg/s.

Table 4.4 summarizes the effect of additional injected traffic in the quality of measurements received at the ADCS computer for a communication bus operating in both nominal (AIT=0 msg/s) and saturated region (6000 and 12000 msg/s for 500 and 1000 kbps, respectively). It shows again that the effect of increasing data rate was bigger than the effect of increasing T_{Sync} on the measurements variance of a saturated bus.

Table 4.4: Variance increase of measurements at the ADCS computer for nominal and saturated communication buses under different data rates and T_{Sync} values

Data rate [kbps]	T_Sync [s]	Variance Increase for the Bus Operating in the Nominal Region	Variance Increase for the Bus Operating in the Saturated Region
500	0.01	0.1%	1%
	0.05	2%	3.5%
1000	0.01	0.05%	18%
	0.05	2%	10%

4.6.4. BUS UTILIZATION BALANCING

The effects of communication channel saturation in the measurement's delay and variability have been characterized using a simulation model. However, for operational purposes, it is also interesting to include mechanisms to deal with data bus saturation once the satellite has been deployed. The key performance indicator for the communication bus is its bus utilization as described in this Chapter. Experimental results presented in the work of Orsel (2016) shows a direct link between the power consumption in the CAN transceiver/controller and the bus utilization which are linked to the data rate and the synchronization period of the bus.

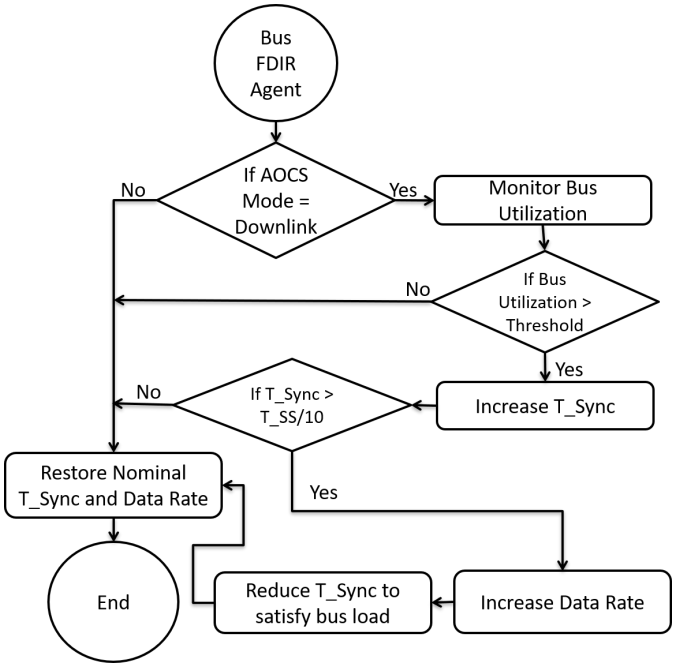


Figure 4.21: Flowchart for an agent-based fault detection and recovery strategy proposed for the mitigation of data bus saturation during AOCS maneuvers.

The bus utilization can be monitored by the CDHS of the satellite by implementing a current sensor in the bus transceiver. Figure 4.21 proposes a flow diagram for the fault detection and recovery agent inside the CDHS. The algorithm described by that flow diagram balances both the data rate and the synchronization period to compensate the additional injected traffic. The agent assumes a CAN controller with flexible data rate capabilities.

4.7. CHAPTER SUMMARY

This Chapter has described in detail the communication stack for MAS according to the FIPA specifications. The first two Sections presented a summary of the principal agent interaction protocols, as well as the details required to implement a message transport service for agents deployed in multiple agent platforms within a satellite.

Section 4.2 described a distributed communication architecture for the implementation of MAS-based software on satellites. That Section suggested the adoption of a CAN-based Message Transport Protocol for enabling the message exchange among multiple platforms of agents distributed along the spacecraft bus. The proposed MTP required the implementation of the Process Data Object protocol on top of CANopen over a linear bus topology configuration.

One of the main contributions of this Chapter was the derivation of an analytical expression for data bus utilization BU in CAN networks under dynamic operations regime. The BU model was tested using a discrete time simulation experiments with two operation scenarios with small satellite missions as a case study.

The simulation results showed that the effect of delays for distributed communication buses operating in the nominal region, do not require to modify the measurement model in the estimation algorithm. The bus utilization in the nominal region showed a linear behavior as a function of the additionally injected traffic. The bus also showed a linear dependency on the network size.

The effect of data rate and bus synchronization period was determined to be inversely proportional to the bus utilization for the nominal region. There is also evidence of a statistically significant difference for data delay in the saturated region compared to the nominal region, leading to a degradation of sensor measurement's performance. The simulations also showed an increase in the measurements variability for saturated channels compared to channels operating in nominal conditions.

Finally, it was suggested that measurements delay induced by bus saturation has a direct effect on the precision of the satellite's attitude state estimation as a result of the change in the sensor measurements variance perceived at the AOCS computer. These findings motivated the proposal a strategy for fault detection, isolation, and recovery based on monitoring the level of bus utilization in the satellite. This metric could be used as an indirect measurement of AOCS performance.

Future research will consider the implementation of the Algorithm in Figure 4.21 as well as a comparison of the results of this implementation with other communication protocols such as I2C to verify and validate the findings. It is essential to consider the newer capabilities of CAN protocols such as flexible data rate for further implementations. The computational burden of delay in the state estimation algorithm can be assessed in future experiments to understand its effects on accuracy.

5

Model-Driven Methodology for Designing Agent-based Software

A theory has only the alternative of being right or wrong.
A model has a third possibility: it may be right, but irrelevant.

Manfred Eigen

Abstract

The objective of this Chapter is defining and describing a methodology for developing agent-based software applications in satellite missions using a model-based approach. The proposed methodology sets out a framework to tailor the modeling process and establishes a robust workflow for analyzing, designing, verifying and refining the implementation of agent-based software for spacecraft subsystems. It also demonstrates the feasibility of using the proposed methodology by describing a case study for the attitude determination and control subsystem. The main advantages of this method include reduced development cycle, separation of concerns and higher adaptability to support advanced algorithms in satellite applications, as well as, built-in fault detection and recovery capabilities.

In recent days, satellite's subsystems development relies more on embedded systems technologies. For highly miniaturized spacecraft, systems developers are attracted to implement their capabilities in software rather than hardware. That is due to implementation flexibility and availability of advanced processing technologies as discussed by Patel and Rajawat (2013).

These new processing technologies bring an opportunity for increasing software's performance, but at the same time a challenge for its design, since it is necessary to enable satellite applications to run as a concurrent and distributed onboard software, where different processing architectures and algorithms can cooperate to achieve the required mission's goal.

For instance, Guruprasad et al. (2016) present a system on chip needed for implementing a software-defined navigation sensor for space missions. Additionally, for current space mission design it is necessary to reduce the development life cycle to fit into more aggressive project schedules and budgets.

Model-Driven Engineering (MDE) offers a framework to describe satellite systems comprehensively. It enables methods and techniques to optimize the requirements analysis and design of onboard software applications. For instance, Rivas et al. (2016) describes the development of a tool-chain for distributed real-time systems that integrates the use of MDE tools to facilitate and accelerate the development process. Model-driven engineering provides as well, with an unified and standardized language to implement models that are easier to read and explain to both decision makers and developers. Regarding complexity, MDE facilitates the handling of implementation details by establishing abstractions concepts and separation of concerns as discussed by Degueule et al. (2017).

According to Paige et al. (2016), one of the main challenges for MDE is to automatically generate efficient and robust code for the targeted hardware platforms, as well as artifacts for physical constraints management. In that sense, the computational model used to produce the code plays a significant role in decoupling domain specific elements from their execution infrastructure.

Agent-based software, in particular, the use of Multi-Agent Systems (MAS), is a new approach adopted in control systems for state estimation as shown by Viel et al. (2017), distributed formation control presented in the work of Kang and Ahn (2016), and several other distributed and concurrent applications with control systems. MAS takes advantage of its flexibility, scalability, and ability to adapt to the dynamic environments to provide a framework for the implementation of distributed artificial intelligence algorithms as discussed by Weiss (1999).

Gascueña et al. (2012) argues that combining model-driven engineering with multi-agents systems offers an opportunity of dealing with the increased complexity of embedded software. However, this requires an extra effort of defining a methodology for end-to-end onboard software design, implementation, and verification. Several methodologies have been documented to develop agent-based software. For example, Prometheus by Padgham and Winikoff (2002)-Padgham et al. (2014), GAAIA by Zambonelli et al. (2003), Ingenias by Pavón and Gómez-Sanz (2003), MaSE by Deloach (2004), and several others offer MDE methodologies for MAS-based software design, but they lack native Fault Detection Isolation and Recovery (FDIR) capabilities.

5.1. MODELING SOFTWARE AS A MULTI-AGENT SYSTEM

Satellite's On-Board Software (OBSW) can be modeled as a MAS, which consists of a collection of autonomous inter-dependent software components known as agents. These agents can be mapped into a set of hardware resources within the onboard computer for their execution. Either the agents or their private behavior functions can rely their execution on software threads as described by Calvaresi et al. (2016).

Additionally, agents can be grouped by their functions or by any other design criteria to form MAS organizations confined into an execution platform. Figure 5.1 depicts how agents interact to achieve a specific goal within the OBSW of the satellite. The MAS platforms can interact with each other using the spacecraft communication bus, as demonstrated in Chapter 4 of this dissertation, which also serves for the agents to interface with sensors and actuators at subsystem's level.

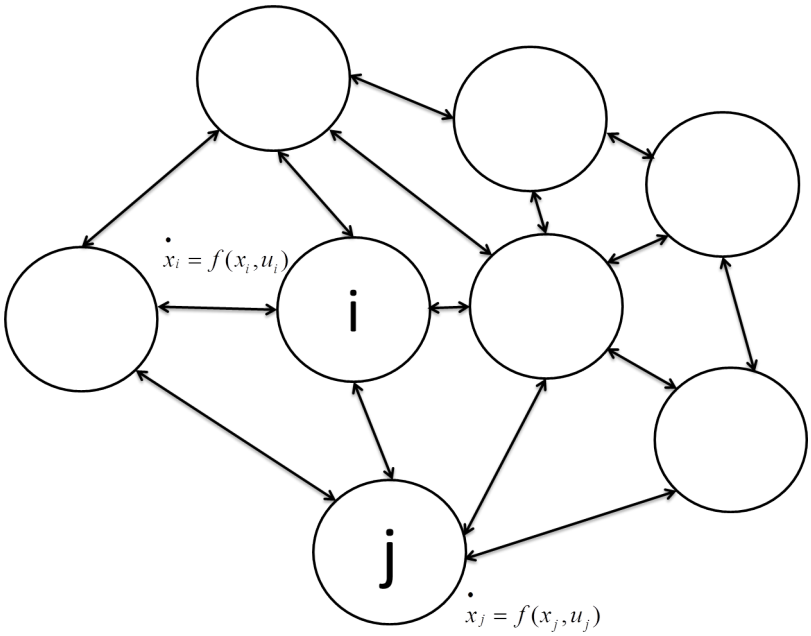


Figure 5.1: MAS-based software topology showing the dynamics of agent's interactions

To model the behavior of the MAS-based software presented in Figure 5.1, consider a network of agents as described by Olfati-Saber et al. (2007a). There, agents and their interactions are represented using a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V} = 1, 2, 3, ..., n$ is a set of nodes corresponding to the agents, and their interactions are described by a set of edges $\mathbf{E}$. The adjacent nodes to agent V_i are represented by $N_i = j \in \mathbf{V} : i, j \in \mathbf{E}$. The dynamics of an agent V_i with state vector $\mathbf{x}_i$ can be described as

$$\dot{\mathbf{x}}_i = f(\mathbf{x}_i, \mathbf{u}_i) \tag{5.1}$$

where, $\mathbf{u}_i$ represents the input vector of agent V_i as a function of its neighbors.

The inputs include sensor and actuator signals that each intelligent agent requires for interacting with other agents or its environment. In discrete time, the collective dynamics of the MAS-based software network in Figure 5.1 can be expressed as

$$\mathbf{x}(k+1) = \mathbf{P}\mathbf{x}(k) \tag{5.2}$$

with $\mathbf{P} = e^{-T\mathbf{L}}$, where T is the update period in the network and $\mathbf{L}$ is the Laplacian matrix of the graph describing the agents network in Figure 5.1. Then, the next challenge is assigning execution resources within the onboard computer for individual agent's execution.

5.1.1. RESOURCE MAPPING STRATEGY

The approach taken to deal with resource allocation in this Chapter uses ideas from the work of Yang et al. (2005). In that work, the main goal was achieving an optimal resource mapping of execution threads for system functions, so that the implementation satisfied a set of design constraints defined by the application, for instance, power consumption or processing time. That work extended its implementation strategy by enabling interaction between their execution threads. For that purpose, it used the agent communication abstraction depicted in Figure 5.2.

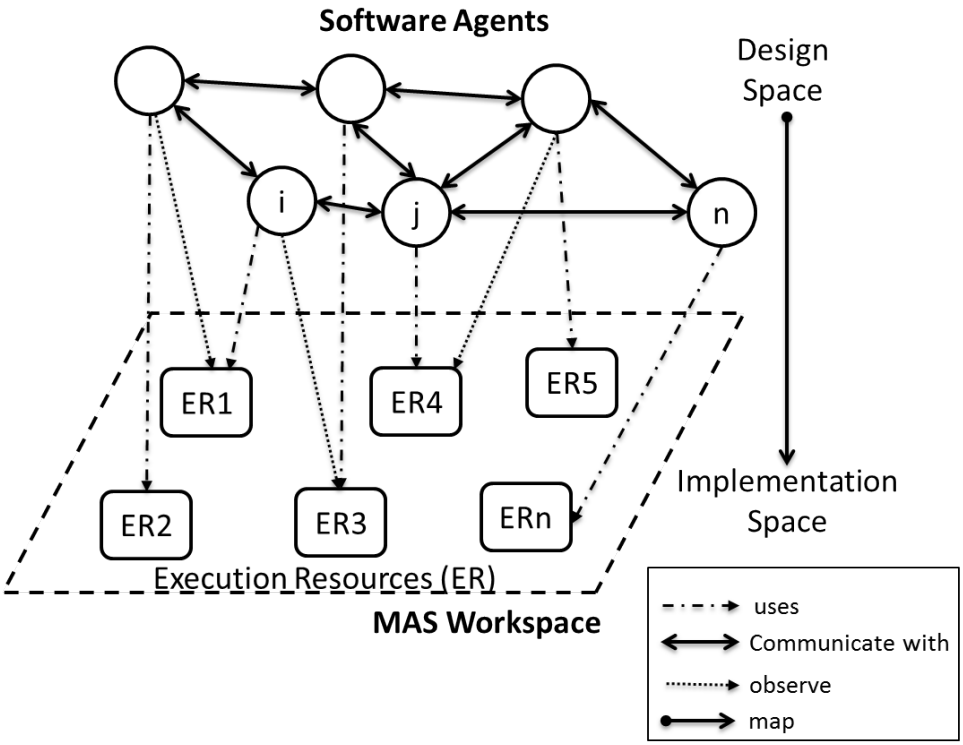


Figure 5.2: Multi-Agent System Resource Mapping Approach with JACA Agents

In that sense, the software application from Figure 5.1 can be implemented by a set of inter-dependent system functions handled by the embedded operating system running on the subsystem's onboard computer. Figure 5.2 also depicts how the MAS-based software design is mapped into the execution infrastructure to enable an embedded MAS workspace. This resource mapping strategy uses the available execution resources within the onboard computer of the satellite to support the execution of a MAS-based onboard software application.

The execution infrastructure can be described as a graph as well, so agent's resource allocation is implemented by establishing a correspondence between agent nodes and their execution resources over time. This mapping approach was defined as the Task-resource scheduling strategy which is presented and discussed in detail by Gorbenko and Popov (2012). In the work of Kwok and Ahmad (1999), the author also reviewed an extensive set of algorithms for resource allocation of static task graphs in parallel execution environments. These algorithms can be also used to compare different agent-resource mapping configurations.

Lately, in the research work of Alghamdi et al. (2017), a mapping algorithm was proposed. It consisted of two-phases, which included a phase for sorting and prioritizing tasks, and a second phase for scheduling allocation. The main advantage of such an approach was that it decoupled the sorting and prioritization, from the scheduling of software threads, which was convenient when working with static graph topologies, as the one assumed on this Chapter.

One of the biggest challenges faced during this research was solving the resource mapping problem under execution resource's contention. That affected the performance, especially on real-time systems that are subject to deadlines for task execution. It was necessary establishing a trade-off process to satisfy the timing requirements of the application and balancing the workload. An iterative design optimized the resource contention, so that it can satisfy missions requirements and constraints.

In summary, in the proposed mapping approach each software agent was linked to one execution thread within the operating system of the onboard computer. In the work of Chan-Zheng (2017), which was developed from the ideas presented in this Chapter, an RTOS thread class was extended to support communication and organization features, so that the graph from execution environment can support the capabilities required for implementation of the MAS-based software application including FDIR capabilities. Now, it is necessary to establish and describe the proposed methodology for developing an end-to-end software application for satellite's missions using the systems model and the mapping concepts discussed above.

5.2. MULTI-AGENT SYSTEMS FOR SATELLITE APPLICATIONS

The primary objective of this section is to present a model-based methodology for designing agent-oriented software for satellite systems. The proposed methodology was called MASSA, which stands for Multi-Agent Systems for Satellite Applications. The goal of MASSA is extending the state-of-the-art features on MAS design methodologies to support native fault detection isolation and recovery (FDIR), as well as enabling tools for extending onboard software development interfaces for FDIR verification and validation that enables a more robust software architecture design and implementation for

satellite subsystems. This section describes how the methodology was conceptualized and its critical processes including its meta-modeling framework, modeling phases, and processes required to implement MASSA methodology in a practical case study.

For that purpose, it was necessary to establish a set of definitions and concepts to support the description process, specifically the modeling structure. An example is shown in the work of da Silva and de Lucena (2007), where a UML for MAS-based modeling language and a set of new meta-modeling concepts were integrated. The main purpose of defining a meta-modelling framework was to standardize the design process, and simplifying its implementation as discussed by Hahn et al. (2009). According to Atkinson and Kuhne (2003), there are four layers defined to support model-driven development. The top layer is the one defining the meta-modelling framework.

MASSA META-MODEL

In order to provide a consistent modeling framework, it was necessary to establish a set of definitions and concepts to support the design of agent-based software, particularly its architecture. Based on the structure described by Atkinson and Kuhne (2003), the top layer is the one defining the meta-modeling framework. This layer defines the concepts required for designing systems by establishing their relationships, processes, and languages. According to da Silva and de Lucena (2007), modeling languages must provide the structural (static and dynamic) aspects of MAS by expressing their characteristics and relationships. For that particular application, an extension of the universal modeling language (UML) was proposed to integrate agent-related concepts into a new meta-modeling framework called MAS modeling language.

The same strategy was followed in the development of MASSA, where the System Modelling Language (SysML) concepts were extended with the purpose of defining a meta-modeling framework to standardize the design process, defining a common vocabulary and simplifying their implementation aspects. A similar application can be found in the work of Antonova and Batchkova (2008), where a similar approach is demonstrated.

In MASSA methodology, the relationship between different elements and concepts is worked out through the MASSA meta-model diagram shown in Figure 5.3, where the central element for the multi-agents system design is the role the agent plays within the virtual organization. Also, a significant factor in the establishment of robust design is the communication infrastructure that enables these interactions.

MASSA's meta-model diagram is based on the comparison and extension of capabilities of methodologies such as ROADMAP by Juan et al. (2002), GAAIA by Zambonelli et al. (2003), Ingenias by Pavón and Gómez-Sanz (2003), and MaSE by Deloach (2004).

The goal of MASSA is to enhance the real-time performance and enabling fault detection, isolation and recovery capabilities by design. That is the reason for adding the FDIR class as an extension of the Behavior class, and observing rules and constraints during the organization process of the satellites onboard software. That is also required for organizational optimization (see Chapter 6) during the resource mapping process.

MASSA focuses on defining a domain specific language (DSL) for satellite software design using an agent-based approach. For that purpose, there was the need to establish a profile that refines the general concepts of SysML and enable their reuse during the

modeling and design process. These stereotypes had to guarantee consistency at syntactic and semantic level with the elements of the MASSA meta-model depicted in Figure 5.3. In the establishment of the MASSA profile, it was necessary to define a set of SysML meta-classes to be extended (e.g., Block, state, port) to support specific agent-based design concepts by applying their derived stereotypes. The classes were grouped into packages for each MASSA methodology phase, where a set of diagrams were implemented for specifying each step in the workflow described in Figure 5.4. The implementation of the MASSA profile was carried out using the Papyrus modeling tools as in Hili et al. (2017).

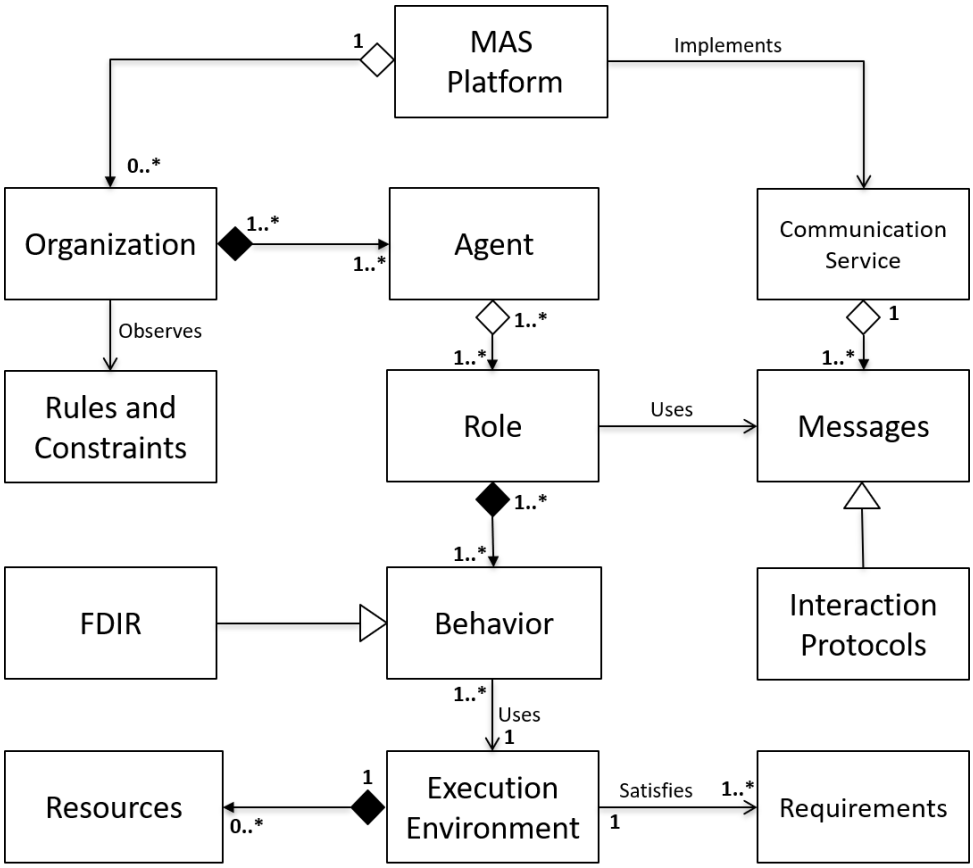


Figure 5.3: Meta-model diagram proposed to implement MASSA methodology

On the communication side, MASSA's meta-model includes three essential elements that enable the proper exchange information between the agents within the platform. These are the agent message, the communication service and the communication infrastructure in the OBC. The meta-model of MASSA in Figure 5.3 provides the foundations for satellite's OBSW developers to model their application using a consistent terminology, but also establishes the domain-specific concepts for their implementation.

MASSA MODELING LANGUAGE AND WORKFLOW

The idea of having a unique language to describe in detail all kinds of systems is not realistic as argued by Fowler (2010). For that reason, system modeling languages and tools have the possibility of tailoring their capabilities to fit into a domain specific application. The way to achieve this level of specialization was by defining customized stereotypes that can be applied to the modeling elements of the MASSA meta-model. For that purpose, there was the need to establish a profile that refined the general concepts of Systems Modeling Language (SysML) and allowed their reuse during the modeling and design process with the MASSA methodology.

MASSA establishes four inter-related activities to sustain the process of designing the onboard software of a satellite. These are a) requirements engineering, b) need and systems analysis, c) software architecture design and d) software architecture verification and validation. The MASSA approach is similar in that sense to modeling methodologies demonstrated in Normand and Exertier (2004) and Kaslow et al. (2015).

In the establishment of the MASSA profile, its was necessary to define a set of metaclasses that can be extended, to support SysML specific concepts by applying the derived stereotypes. The classes were grouped into packages for each MASSA methodology phase, where a set of diagrams was implemented for specifying each step in the workflow presented in Figure 5.4.

Figure 5.4 shows the proposed workflow for the MASSA methodology. Along their phases there is a process for requirements management, that supports all the phases in parallel. Through these activities, MASSA intended to respond to the following questions:

1. What is the mission that the users of the system need to achieve?
2. What does the system need to provide to the users to achieve that mission?
3. How does the system can provide the required capabilities?
4. How to allocate the available resources with the capabilities needed?
5. How to provide feedback to the design process to make system software architecture more robust to errors and failures?

It is also important to discuss how to complete each phase on the MASSA's workflow. The analysis phase responds to the first and second question. The first question is addressed by understanding the operational conditions of and performing a needs analysis. In that analysis, both the actors and their required capabilities are studied from the user's viewpoint. For that purpose, Use Case scenarios are documented, as well as, operational workflows for the system. These activities enable the identification of capabilities for the system to be designed. The second question is addressed by performing a systems analysis. It allows using the identified actors and their required capabilities to establish roles, functions and the data flow required for satisfying the capabilities needed. MASSA also defines optimization loops to improve efficiency in the operational workflows according to its functional decomposition.

Questions 3 and 4 are addressed during the design phase where both a logical and a physical architecture are specified and iterated by making use of the optimization loop

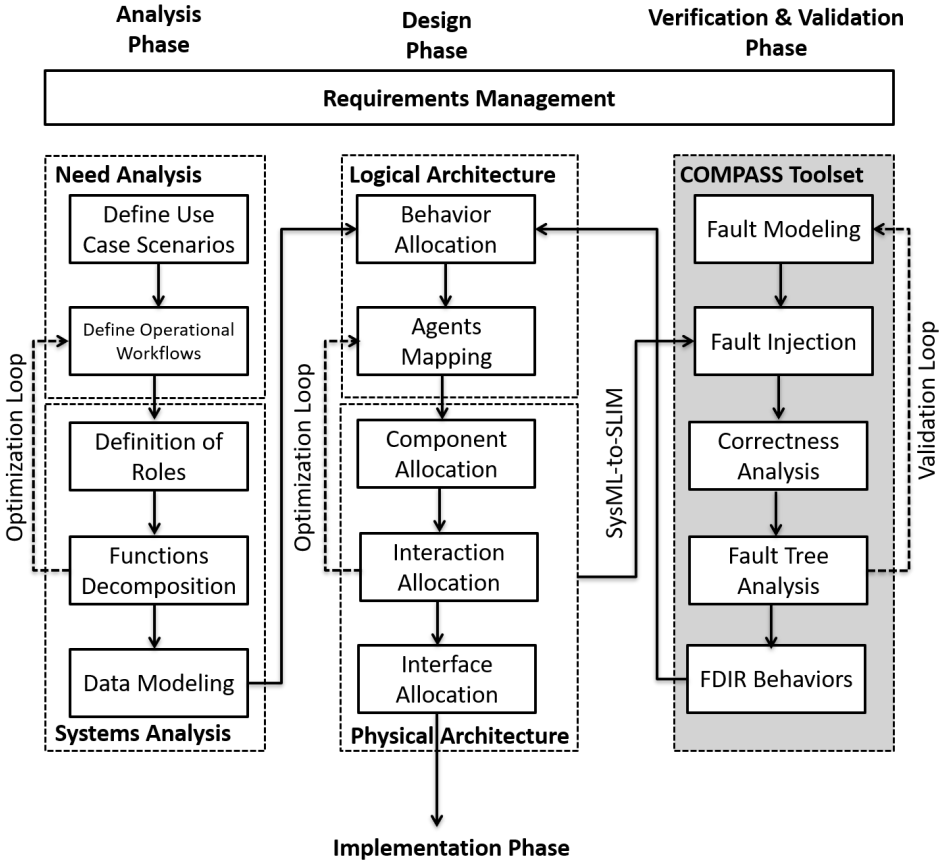


Figure 5.4: MASSA Workflow Diagram

between physical and logical architecture. During the design phase, the behaviors derived from the functions are subsequently allocated into agents that interact within a logical container in the execution platform (i.e., onboard computer). Also in the design phase, the interaction between agents is defined by implementing agent acquaintance's identification techniques oriented to specify the type of interaction protocol needed to satisfy the data models identified in the analysis phase. This activity is called the agent mapping procedure.

Additionally, an interaction scheme is allocated with its respective communication interfaces for the agents to interact with the systems environment and other subsystems components. There is an optimization loop intended to minimize the communication overhead by allocating agents in organizations that maximize the use of shared resources. The optimization algorithms for organizational optimization are introduced and developed in Chapter 6 as part of this dissertation.

Finally, there is the FDIR verification and validation phase intended to proactively identify and correct fault mechanism that causes failures in the system due to software design gaps. In MASSA a fault injection is modeled by adding properties to the nominal SysML model.

Faults are understood as the manifestation of errors in the modeling, design or implementation of software features of the system, while failures are considered as a deviation of the software performance with respect to its requirements.

The approach followed in this phase is based on the work presented in Bozzano et al. (2009) using the COMPASS toolset for model checking. It assumes the prior knowledge of fault mechanisms and their impact on systems performance. These mechanisms are translated into design requirements that are later verified and validated using the model checking tools.

The implementation of the verification phase required a model-to-model transformation to take the physical architecture generated by MASSA in SysML and translating it to System-Level Integrated Modeling language (SLIM) for its verification with the model checking tools. First, a set of nominal behaviors are identified in the design phase and allocated to agents and their organizations. Then, a set of faulty behaviors are specified and implemented using the error models specified as requirements.

The error models are used to implement fault models. These errors are described separately to guarantee separations of concerns during the dependability and correctness analysis. In the context of MASSA, correctness is focused on the functional aspects, meaning that for each function the input-output results are compared to pre-established patterns to test the correct operation. Also, errors are linked to the specific components configuration, so the model checking tool, in this case, COMPASS can generate fault trees to analyze the occurrence of a failure due to multiple error sources. Failure mode effect analysis (FMEAs) are used as inputs to generate functional requirements for the software architecture design.

Faults require to be combined with the nominal behaviors, which is done in the fault injection step. That action triggers the correctness analysis that produces a fault tree analysis that is later combined with other analysis (for example safety or dependability) to allow identifying FDIR behaviors. FDIR behavior is derived from analyzing the cause of a fault/failure and establishing a recovery procedure that mitigates errors leading to the failure. Once the FDIR behaviors are identified and described, they are integrated with the nominal behaviors, so that the logical architecture is updated. In the FDIR verification and validation phase, there is also a validation loop to ensure that the signatures observed are consistent with the fault models so that the fault models can be validated. The implementation of the FDIR phase was handled as an spin off work documented in the Master's thesis of Joost Van der Gaag (2017).

Concerning the implementation phase of MASSA, for now, this work does not specify any restriction. However, in the future, the idea is using the MASSA profile created in Papyrus for automatic code generation.

As of now, once the logical and physical architecture are refined the implementation of the resulting architecture is done manually using customized Java or C++ libraries. The following section will elaborate on a case study where the selected steps of the proposed MASSA methodology are illustrated by example to illustrate its feasibility.

5.3. ADCS CASE STUDY

This section elaborates on a case study where each phase and step of the proposed MASSA workflow from Figure 5.4 is implemented. Within a satellite, the attitude determination and control subsystem (ADCS) requires the implementation of complex algorithms for guidance navigation and control. The most common algorithms for attitude estimation are based on the Kalman filter, in particular, the extended Kalman Filter (EKF) that account for non-linear measurement models as discussed by Lefferts et al. (1982). In Appendix C more details about its implementation are explained.

Depending on the mission needs, the satellite can be equipped with sun sensors, magnetometers, inertial measurement units (IMU) and GPS receivers. Figure 5.5 shows the architecture for the ADCS subsystem that is used to develop this case study. It illustrates the interface between hardware and software components required for designing an MAS-based attitude estimation algorithm.

For simplicity, it assumes both the sensors and the command and data handling computer (CDHS) are connected to the spacecraft data bus. Due to the need of having a precise orbit estimation capability, the satellite is equipped with two GPS receiver modules to measure position and estimating velocity. The following section describes the physical model used to develop the case study implemented with MASSA methodology.

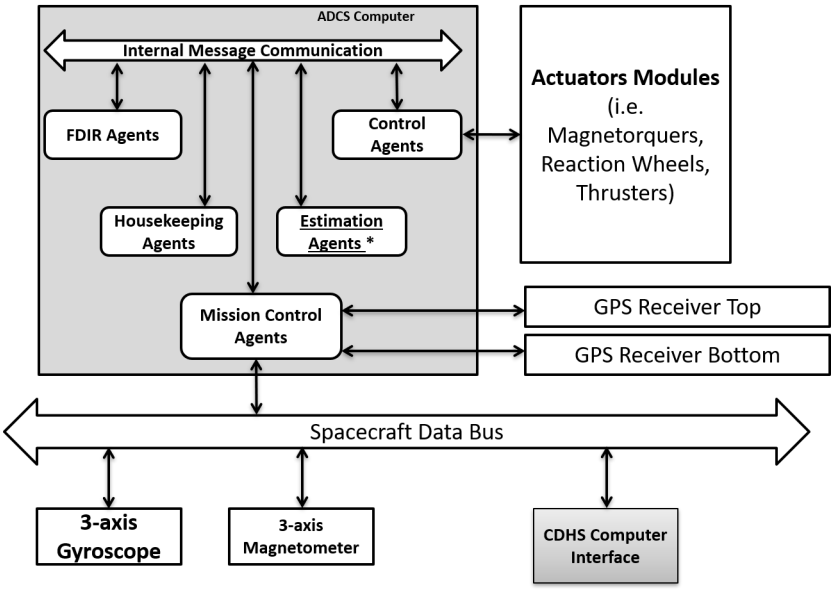


Figure 5.5: Physical Architecture used for the ADCS case study

5.3.1. ADCS PHYSICAL MODELING

The goal of this case study is to show the feasibility of MASSA methodology to design a multi-agent based algorithm for satellite's attitude estimation using the workflow from Figure 5.4. The attitude state for the satellite is described using the quaternion $\mathbf{q}(t)$ notation. The estimation algorithm was implemented using as reference the work presented in the work of Zeng et al. (2014).

The attitude state of the satellite is represented over time as $\mathbf{x}(t) = [\mathbf{e}^T, q_4]$, where $\mathbf{e} = [q_1, q_2, q_3]$ and q_4 is the scalar component of the quaternion in the orbital coordinate system (see Chapter 2 for more details on reference frames). The angular motion of the satellite is described by the attitude kinematic in (5.3), which is consistent with the multi-agent model presented in (5.1). In (5.3), $\mathbf{\Omega}[\boldsymbol{\omega}_B]$ is a 4x4 skew symmetric matrix defined as a function of the satellite body angular velocity $\boldsymbol{\omega}_B = [p, q, r]^T$ which is relative to orbital coordinate system as

$$\dot{\mathbf{x}}(t) = \mathbf{\Omega}[\boldsymbol{\omega}_B]\mathbf{x}(t) \tag{5.3}$$

$$\mathbf{\Omega}[\boldsymbol{\omega}_B] = \frac{1}{2}\begin{bmatrix} \boldsymbol{\omega}_B \times & \boldsymbol{\omega}_B \\ -\boldsymbol{\omega}_B^T & \mathbf{0} \end{bmatrix} \tag{5.4}$$

The operator $\boldsymbol{\omega}_B \times$ represents the cross product of the angular velocity as

$$\boldsymbol{\omega}_B \times = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{5.5}$$

To implement the model in (5.3), it has to be in discrete-time considering the period T_S for the numerical integration. The resulting discrete-time model for the quaternion model with initial conditions $\mathbf{x}_0 = \mathbf{x}(0)$ is described as

$$\mathbf{x}_{k+1} = e^{\mathbf{\Omega}_k T_S}\mathbf{x}_k \tag{5.6}$$

To obtain a proper attitude propagation, the angular velocity measurements had to be transformed from the body frame to the orbital frame as

$$\boldsymbol{\omega}_B = \boldsymbol{\omega} - \mathbf{C}(\mathbf{q})\boldsymbol{\omega}_0 \tag{5.7}$$

where, $\mathbf{C}(\mathbf{q})$ is the attitude transformation matrix from body coordinate systems to orbital coordinate system, and $\boldsymbol{\omega}_0$ can be obtained from satellite's orbital parameters and $\boldsymbol{\omega}$ is body inertial angular velocity.

The measurements obtained from the GPS were used to calculate the satellite position and velocity $\mathbf{x}_O$ that are needed in the attitude estimator for coordinate systems transformation. The measurements from the magnetometer are used to correct the attitude values propagated using (5.3). The magnetometer measurement model used for the case study is presented as function of the current position and velocity of the satellite determined using the GPS measurements as

$$\mathbf{B}_B = \mathbf{C}(\mathbf{q})\mathbf{B}_0(\mathbf{x}_{orb}) + \mathbf{v}_B \tag{5.8}$$

where, $\mathbf{B}_B$ represents the actual measured value of magnetometer, $\mathbf{B}_0(\mathbf{x}_O)$ is the magnetic field vector in the orbital coordinate system and $\mathbf{v}_B$ is the measurement noise.

5.3.2. MASSA: ANALYSIS PHASE

Once the physical model and the ADCS reference architecture were specified, MASSA methodology was applied to analyze the ADCS subsystem following the steps established in Figure 5.4. The two most relevant diagrams are shown in this section to illustrate the results of the proposed MASSA's analysis phase. They were synthesized using SysML language using Papyrus modeling artifacts. Firstly, a set of use cases were identified for the ADCS software in Figure 5.6. The second diagram in Figure 5.7, illustrates the ADCS subsystem breakdown, which shows how functions and behaviors are linked for further allocation in the design phase.

Figure 5.6 shows the use case diagram created to identify the OBSW requirements for the ADCS case study. In this diagram, there are two actors identified, that represents the external interaction with the system. These are the satellite operators and the CDHS computer. Within the system of interest, the use case diagram allows identifying three main software use cases for telemetry collection, maneuver execution, and fault detection and recovery. These main use cases are associated with secondary use cases to achieve their overall need of the mission.

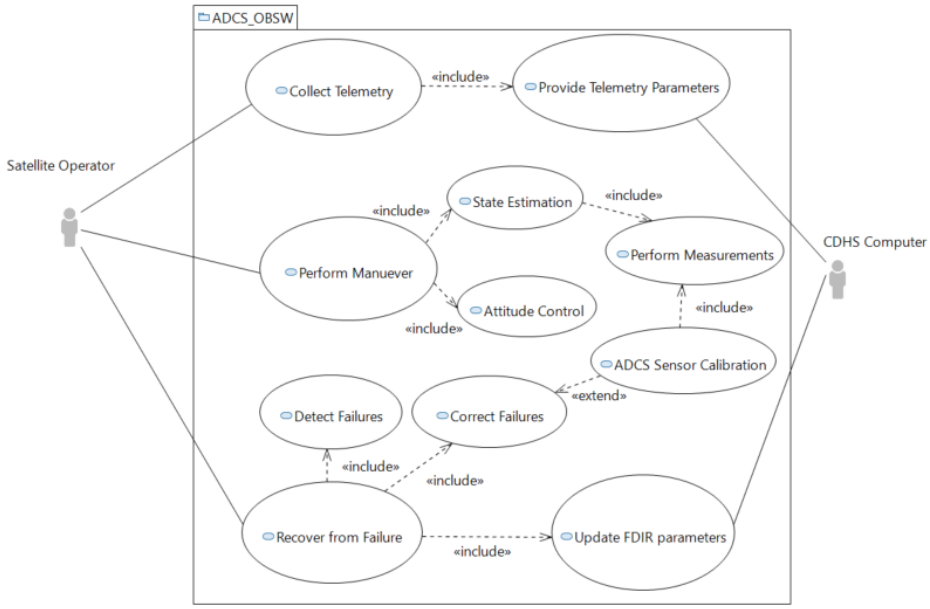


Figure 5.6: Use Case Diagram for the ADCS software in a Satellite Mission

The focus was put on the MAS-based software development in the ADCS computer package that included sub command execution, telemetry management, fault detection and recovery, attitude estimation and control. Special attention was given to the most

complex use case that was attitude estimation capability. Following the MASSA methodology, Figure 5.7 presents the steps related to roles, functional, and behavior identification. For that diagram, the use cases were used as inputs input, as well as the ADCS mission requirements. In that tree, the behaviors were linked with their parent role, and then they were broken down into functions that are implemented separately as agents running within the MAS execution environment.

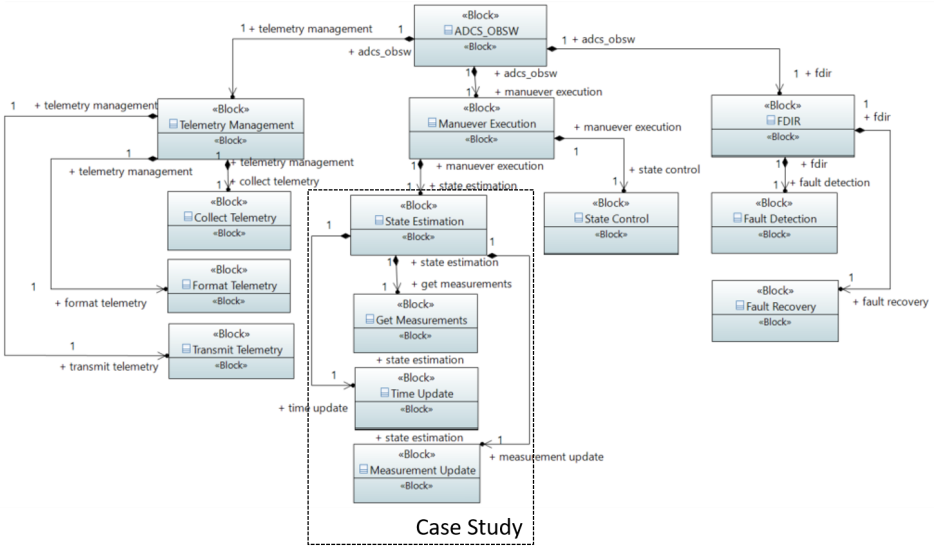


Figure 5.7: Function Decomposition Diagram for the ADCS Software in a Satellite Mission

As a result of the analysis phase, it was possible the identification of five main roles. These are mission control role, in charge of the interface management with other subsystems in the satellite. Housekeeping role to collect and report data about the performance and the health of the satellite to the command and data handling subsystem. FDIR role in charge of hardware and software redundancy procedures to guarantee satellites operation's safety; attitude control role that has the duty to correct the orientation of the spacecraft and finally the attitude estimation that is the focus of this case study.

Then, the attitude estimation role was broken down into behaviors that required the implementation of a set of independent software functions. The necessary functions identified were (a) EKF filter initialization, (b) data acquisition, (c) parsing measurement data, (d) update transition matrix, (e) update rotation matrix, (f) update Jacobian matrix, (h) update measurement vector and (i) update noise matrices.

Four main behaviors were synthesized for the estimation role using these functions:

- Get Measurement: This is a cyclic behavior that enables the interface and collection of the measurement data from the sensors. The data from the gyroscope is used to propagate the state vector whereas the magnetometer data is used to correct the attitude estimation in real-time.

- Compute Kalman Filter: This behavior implements a finite state machine for controlling the execution of the extended Kalman filter (EKF) algorithm. There are 5 states for the implementation of the EKF behavior: (1) Data acquisition, (2) Data formatting, (3) Prior state computation, (4) Kalman Gain computation and (5) Posterior state computation.
- Compute state: this behavior interacts with the compute Kalman filter to either compute the prior or posterior state vector.
- Compute co-variance matrix:this behavior interacts with the compute Kalman filter to either compute the prior or posterior co-variance matrix.

In the design phase these behaviors are allocated and organized to satisfy the ADCS subsystem's needs.

5.3.3. MASSA: DESIGN PHASE

As discussed in the analysis phase, behaviors were broken down into functions that required to be executed in a sequence. The agent allocation followed a one-on-one mapping approach discussed in Subsection 5.1.1 to preserve the independence in the execution of these behaviors, and maximizing code reuse. Each behavior of the estimation algorithm was mapped into one software agent. These agents were deployed in a single Multi-Agent Systems logical container, which represented the execution platform (OBC) of the ADCS subsystem.

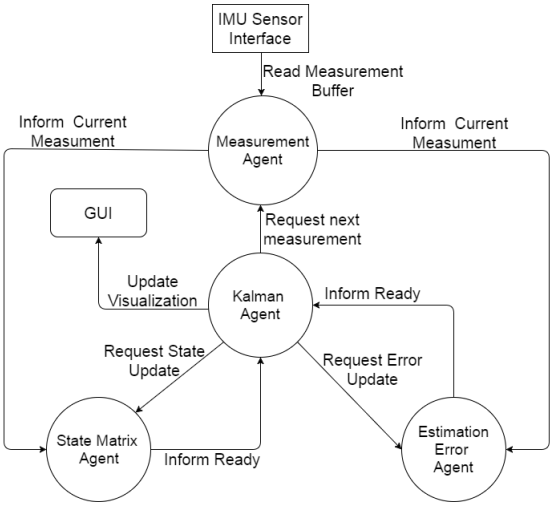


Figure 5.8: Agent Interaction Diagram for the EKF Estimation Algorithm

The most important step in the design phase is the allocation of interaction and communication for the multi-agents system as argued by Norman et al. (1996). For doing that, an agent acquaintance and relationship process was performed for identifying communication dependencies and data flow sequences. Also, it was essential to define a proper message structure and semantic needed for lower level implementation.

Figure 5.8 summarizes the communication allocation process for the attitude estimation functionality within the ADCS onboard software with the Extended Kalman Filter algorithm. This diagram shows the flow of data from sensors to the estimation, and then to the visualization tool used to verify the implementation. Also the diagram shows the centralized architecture used for the implementation of the estimation role. It was done this way to be consistent with the MASSA meta-model presented in Figure 5.3, where the role is the core of the MAS-based software conceptualization.

The final step in the design phase was the interface allocation. For the case study there were three main interfaces present in the estimation algorithm. One was needed to access and configuring sensors. The second interface provided the attitude control agents with the current state estimation, and the third interface was used to visualize the results of the EKF algorithm and verifying its performance.

From this point, the MASSA workflow establishes two ways to continue. Either the developers moved forward to the implementation and deployment phase, or they continue to the model-based verification phase discussed in details in the following section.

5.3.4. MASSA: VERIFICATION PHASE

The cost of fault detection increases as the project development advances. According to Feiler (2010), only 3.5% of faults are detected and corrected during the design phase of safety-critical embedded systems. This effect is negligible compared to the 70% of faults that are generated at the design process. The way to address the efficiency of fault detection during the design phase is by adopting continuous verification and validation. This is the reason why MASSA methodology includes this philosophy as a key element of its workflow.

As part of this work, a complementary verification methodology called SysML to SLIM Transformation Methodology (SSTM) was developed to facilitate the verification process. According to der Gaag (2017), a tailored SysML profile is accompanied by a model conversion tool for transforming the system model from SysML language to System-Level Integrated Modeling (SLIM) language for its analysis with the Correctness, Modeling and Performance of Aerospace Systems (COMPASS) tool set approach presented by Bozzano et al. (2009). The results of the verification are not shown here, but they could be seen in the work of der Gaag (2017).

The SSTM workflow is compatible with the MASSA methodology, and it provides information for agent's behavior refinement. The system model was complemented with a set of fault models that are connected for identifying the probabilities of function failure in the COMPASS toolset, then visualized as a fault tree diagram. COMPASS also allows the verification of the system model correctness and it enables simulations to visualize the failures evolution over time.

In the case study for the ADCS subsystem, errors were introduced in the communication with CDHS subsystem, ADCS sensor blackouts, and actuator saturation. After that, particular attention was given to faults caused by systematic measurement degradation, which was root-caused to drift and bias in the data used to propagate attitude's state. The output of this analysis was then used for model and MAS architecture improvement.

MASSA: MODEL AND ARCHITECTURE IMPROVEMENTS

MASSA establishes a feedback loop between the verification phase and the design phase to enable improvements on the behavior allocation shown in Figure 5.4. Using this logic, for this case study the estimation model was updated to include an additional state vector for the estimating the gyroscope bias to mitigate potential errors during run-time.

Expression in (5.9) updates the state estimation model from (5.6) to include the gyroscope drift bias vector $\boldsymbol{b}_k^g$ and its noise $\boldsymbol{v}_k^g$ as part of their state vector. The goal was to use the estimated gyro drift and bias to compensate the deviation in the angular velocity $\boldsymbol{\omega}$ over time, so that the EKF implementation can reduce the estimation error. That is implemented as a behaviour in the FDIR agent in Figure 5.9.

$$\mathbf{x}_{k+1} = \begin{bmatrix} e^{\boldsymbol{\Omega_k} T_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{3\times 1} \end{bmatrix} \begin{bmatrix} \mathbf{q}_k \\ \mathbf{b}_k^g \end{bmatrix} + \begin{bmatrix} \mathbf{v}_k^q \\ \mathbf{v}_k^g \end{bmatrix} \tag{5.9}$$

A new agent (FDIR Agent) was implemented to include a behavior to calibrate the sensors when there is a failure detected, as well as, the implementation of a memory buffer to hold the latest measurements, so in the case of a sensor blackout, an average of the latest 30 measurements can be used by the estimator. Besides the addition of the FDIR Agent, the architecture of the estimator software shown in Figure 5.8 kept similar. On the measurement model, there was not any update required. The gyroscope data continued to be used for the prediction functions, as the magnetometer continued to be used to correct the orientation by providing the magnetic field vector as a function of satellite's position.

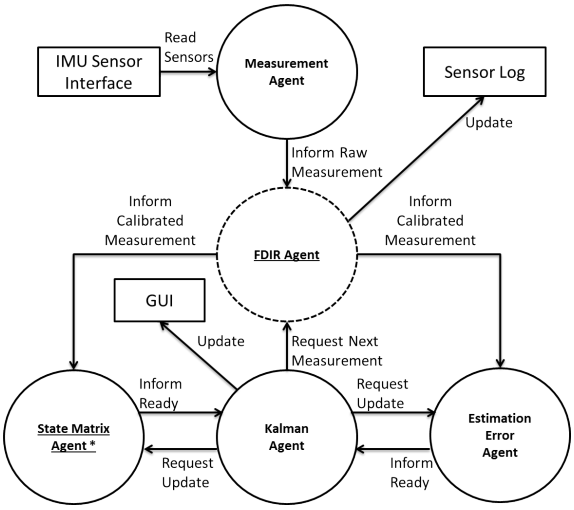


Figure 5.9: Improved Agent Interaction Diagram for the EKF Estimation Algorithm

The next subsection shows the results obtained from the implementation of both estimators (before and after the verification phase). The results are used for highlighting the benefits of earlier verification during the implementation phase using MASSA's approach.

5.3.5. RESULTS ANALYSIS FOR THE ADCS CASE STUDY

This section presents the results of the implementation of attitude estimation algorithms for the ADCS case study described by the agent interaction diagrams in Figure 5.8 and Figure 5.9. The goal of this experiment was to show the effects that the verification and validation phase of MASSA workflow has on the performance of the implementation. Also, it was intended to show the use of the verification and validation feedback on the design optimization described in Figure 5.4. The experiment was carried out by implementing the ADCS physical architecture from Figure 5.5 in a fixed location to avoid reference frame transformations. Then, simple attitude maneuvers to test the estimation algorithm accuracy were executed. Figure 5.10 shows the experimental setup as a block diagram.

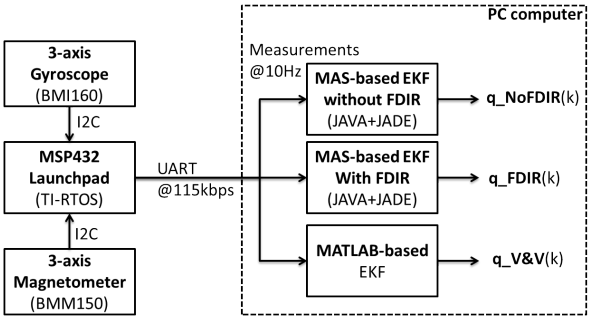


Figure 5.10: Experimental Setup Testbench for ADCS case study

In Figure 5.10, the data from the sensors is collected at a 10 Hz frequency using an embedded computer running the Texas Instrument real-time operating system (TI-RTOS). The sensor information (3-axis gyroscope and magnetometer) was sent using the Universal Asynchronous Receiver-Transmitter (UART) known as the serial port to the computer implementing the EKF algorithms with the Java Agent Development Framework (JADE) library and running on top of a JAVA virtual machine. There are three variants for the EKF algorithm. One (top block) to implement the architecture from Figure 5.8, the second (medium block) to implement the architecture from Figure 5.9, and the bottom block is used for numerical verification of the results.

The experiments conducted include collecting a static maneuver where the sensors are kept fixed in a position to demonstrate the effects from cumulative faults such as sensor drift and bias. Other maneuvers like pointing and returning to rest position were tested to show pointing maneuvers capabilities, but not included in this section. The main experiment carried to test the effect of including the feedback from the verification phase of MASSA was the static maneuver. For that purpose, the experimental test-bench from the Figure 5.10 was used keeping the sensors in a fixed known position.

Five trials of 10 minutes duration were executed. First, the numerical consistency of the MAS-based EKF with respect to a reference MATLAB ^{TM} implementation was verified. Then, a comparison between the performance of the pre-verification architecture with respect to the improved EKF estimation algorithm was performed, including the Fault Detection, Isolation and Recovery (FDIR) Agent.

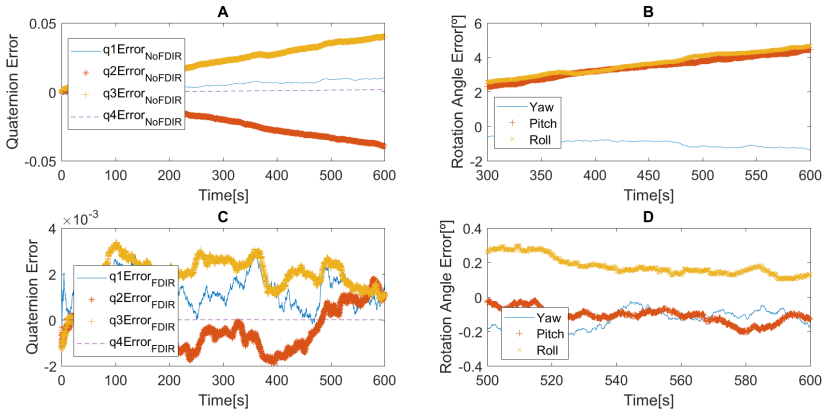


Figure 5.11: Implementation Results: (A) Quaternion Error for MAS-based EKF without FDIR. (B) Rotation Angle Error for MAS-based EKF without FDIR. (C) Quaternion Error for MAS-based EKF with FDIR. (D) Rotation Angle Error for MAS-based EKF with FDIR.

The results of the experiment are summarized in Figure 5.11. There, the true attitude quaternion was fixed to a static position represented with the quaternion value $\mathbf{q}_{True} = (0,0,0,1)$. Then, the sensor data was collected, and post-processed for current orientation visualization. The graphs labeled as A and C show the estimation error based on the quaternion measurement for the pre-verification and post-verification cases, respectively. This error was calculated using the expressions: $\mathbf{q}_{Error} = \mathbf{q}_{True} - \mathbf{q}_{NoFDIR}$ and $\mathbf{q}_{Error} = \mathbf{q}_{True} - \mathbf{q}_{FDIR}$ for both cases, respectively. The graphs labeled as B and D are the rotation angle error for both cases. They are calculated using the quaternion obtained from the EKF.

By analyzing the results of the EKF with no FDIR agent implemented (A and B), it is clear that there is a drift on the estimated orientation for the static case. This is due to sensor measurement faults that are propagated in the EKF. After 10 minutes of operation, the rotation angle error is bigger than 4 degrees, which is not compliant with the ADCS design requirements that is usually less than one degree.

On the other hand, once the FDIR agent was implemented, the quaternion error in graph C is reduced about 10 times compared to graph A, and it does not show any time-dependant behaviour. That is translated into a rotation error angle with absolute value lower than 0.4 degrees (graph D), which satisfies the ADCS design requirements. The experiment has shown the value of early verification and validation in the performance of the algorithm implementation, which verifies the initial hypothesis that earlier verification improves system performance. The main benefit is that flexibility of MAS-based EKF implementation allowed improving the performance with minimal changes.

This kind of cumulative errors can cause non-desired performance issues on the pointing accuracy. Also over time, the pointing accuracy can degrade to the point of causing a failure to achieve the mission targets. That is where having an integrated FDIR approach shows its value for space mission operations.

With respect to the MASSA implementation, the case study allowed to exercise the

workflow proposed in Figure 5.4 including the analysis and design phases. It allowed to improve the ADCS performance by providing feedback to its design process and integrating native FDIR features.

5.4. PROPOSED MASSA VALIDATION STRATEGY

So far, the case study shown that the proposed workflow is feasible and provides benefits in term of performance for agent-based OBSW design and implementation. However, for strictness purposes it is necessary at least to provide a view on how MASSA methodology can be validated and compared against well-established modeling methods used in industry for model-based software design (e.g Arcadia/Capella).

According to Boehm (1984), validation activities must check that software products are complete, consistent, feasible and testable. Completeness refers to the ability of a software product to link the features provided with the list of needs identified during the analysis phase. Also it requires to check that all the functions are identified and described. Consistency is defined as a characteristic of the produced software to comply with standards and specifications in the model. For example, for a EKF, consistency on the input/output matrices sizes requires to be checked during the verification phase. Consistency also requires implementing traceability of the process followed. Regarding feasibility, validation activities requires checking that all functions can be implemented, but additionally requires an assessment of reliability, maintainability and scalability. Finally, testing is checked defining clear metrics that enables determining that requirements are achieved according the customer needs.

Table 5.1 shows how these validation criteria are enforced in each of the methodology phases of Multi-Agents Systems for Satellite Applications. A tool or set of tools for each validation criterion and methodology phase are specified so that the criterion can be enforced as part of the software development activities.

This table also shows how to connect MASSA's phases to other artifacts for implementation such as class diagrams and fault tree diagrams. The consistency of the model is shown using simulations that consider the physical model described in subsection 5.3.1 and implementation parameters, including sensor's data sheets.

Table 5.1: Validation Matrix proposed for MASSA Methodology

	Validation Criteria			
MASSA Phase	**Completeness**	**Consistency**	**Feasibility**	**Testeability**
Analysis	Use Case Scenario Diagrams and Function Decomposition Diagrams	Workflow Specification (Figure 5.4)	Case Study Implementation (Reference Applications)	Requirements Metric Assessment
Design	Agent Interaction Diagrams			Systems Budget Analysis
Verification	Fault Tree Diagrams	Simulation Models		Correctness Analysis
Implementation	Class Diagrams	Detailed Operation Scenarios		Unit testing Framework

5.5. CHAPTER SUMMARY

This Chapter has proposed a meta-model, a modeling methodology and a case study to demonstrate the feasibility of MAS-based based software design for developing and implementing the onboard software of a satellite. The Chapter has described a meta-model that extends the fault detection capabilities of MAS-based software components for improving its implementations performance. Also, it defined a workflow for the implementation of the proposed meta-model.

This Chapter has described an iterative model-based design process that enhances the robustness of the onboard software implementation for satellite applications. This work also has shown a case study that demonstrates the feasibility and applicability of MASSA methodology for designing software for critical satellite subsystems. The use of the verification and validation feedback allowed to enhance software performance by extending the MAS-based software architecture with new functionalities. A validation approach was described according to four main criteria that can be used to show the robustness of the proposed methodology.

The Chapter has shown the value of the multi-agents systems technology to enable a faster and more robust software design process by adopting earlier verification to refine the flight software modeling and implementation.

The case study can be extended in the future to other algorithms within the ADCS subsystem such as the control and the telemetry management, as well as, to other subsystems for instance the command and data handling. The next Chapter focuses on organizational optimization of MAS-based software architectures to fit into the constrained resources of the onboard computer of a satellite.

6

ORGANIZATIONAL OPTIMIZATION OF MULTI-AGENT BASED SOFTWARE

"It is during our darkest moments that we must focus to see the light"

Aristotle Onassis

Abstract

The optimal architecture of a multi-agent-based software is determined by its agent's organization and the modeling techniques employed to describe their interactions. Robust and fast consensus are indicators of success in multi-agent-based algorithms. Reliability and cost of communication can be used to constrain the solution domain in the organizational optimization of agent-based software. Meta-heuristic methods are computationally viable to find a solution of optimization problems within a reasonable number of iterations. This Chapter shows the use of methods based on random approaches such as genetic algorithms and particle swarm optimization for finding a quasi-optimal solution to solve the design of interaction topologies for multi-agent-based algorithms software implemented on satellite systems. The results show that compared to brute force algorithms the proposed approach is up to 200 times faster and allows to tailor specific constraints for different satellite mission scenarios that can be used to optimize reliability during spacecraft operations.

The optimal architecture of a system is mainly determined by the structural configuration of its components, which determines their interfaces. It is necessary to have a clear understanding of the system requirements and constraints to achieve the best performance. In particular, for Multi-Agent Systems (MAS), reliability in the communication process is one of the key challenges for its implementation's success as discussed in Chapter 4. The design of software component's interactions demands a systematic approach to find a solution that both satisfies the specific needs of the application but also addresses resource allocation during its implementation phase. Modeling methods employed to describe MAS can be applied to optimize their implementation as shown by Olfati-Saber et al. (2007b). They describe the systems functionality as a dynamic network of agents looking to reach consensus. In order to find an agreement, agents must implement an interaction topology to share their knowledge across the entire community of agents.

The cost of communication is a limiting factor in the organization's topology of multi-agent systems used for onboard software of space systems. For that purpose, this Chapter proposes to define the Cost of Communication Matrix (CCM) to quantify the effort needed to transport information from one point to another in the multi-agent system graph, given a known probability of interaction. The communication effort can be measured in power, time or any other resource unit. The CCM shall then be used as an input to the topological optimization process to constrain the solution space.

The problem of the optimal organization of distributed systems is receiving more attention recently. For instance, Moghaddam and Jovanovic (2017) describe the optimal topology design of corresponding edge weights for undirected consensus networks using a convex optimization method that balances the performance of stochastically-forced networks. Liu and Iwamura (2000) describe topological optimization models for networks with multiple reliability goals using dependent-chance goal-oriented programming to search for an optimal topology in constrained networks.

Combinatorial optimization is also intended to find the best arrangement of objects in a discrete space, so that they satisfy a set of requirements. According to Lawler (1976), the best way to represent optimization problems in discrete solution spaces is using graphs. Randomized optimization algorithms have become popular to solve combinatorial optimization problems due to the increased availability of computing power. That processing availability reduces the time to find a feasible solution. However, drawbacks have to be addressed when selecting a search algorithm family, such as the tuning process and the implementation complexity. Recently, randomized algorithms have been used to optimize connection weights in neural networks, as shown by Aljarah et al. (2018).

The purpose of this Chapter is to identify and implement an algorithm for the optimization of agents interactions in MAS-based software for satellite applications using randomized optimization techniques. This is relevant for onboard software designers, since it enables a tool for design exploration that takes into account functional requirements and implementation constraints, for instance, the power budget. This Chapter also describes the total cost of communication in a MAS-based software as a function of an agent interaction matrix and the importance of their links (weights). This modeling approach enables novel mechanisms to describe communication constraints based

on fixed topological structures such as hierarchical, team and federated communication design patterns. Then, it proposes the use of randomized optimization methods for minimizing the total cost of communication needed to satisfy the software configurations of space missions. The Attitude Determination and Control Subsystem (ADCS) case study was used to demonstrate its feasibility for the onboard software of spacecraft.

6.1. ORGANIZATIONAL STRUCTURES FOR AGENTS

Firstly, it is important to motivate the need for organizing Multi-Agent Systems. According to Ferber et al. (2003), in the context of MAS, an organization provides a framework for operation and interaction through the definition of roles, behavioral expectations and authority relationships. This is a key element that must be considered in the modeling and design of software systems using a MAS-based approaches.

The process of establishing the organization of Multi-Agent Systems responds to physical and logical aspects during its implementation phase. Physical aspects refer to the structural elements that enable the partitioning of tasks within the system into software functions. Logical aspects are more abstract and they make reference to the role allocation of agents (features) required during the run-time. A third element to consider is the dynamics of these agents, which is key for the correct and optimal allocation of satellite resources. The manifestation of agents dynamics is described by their interactions patterns. That was the main characteristic of MAS-based architectures addressed in this Chapter.

The process of designing an organization requires defining a logical structure for achieving a goal. In the context of onboard software, an organization encompasses the logical layout of the software components needed to satisfy the missions requirements and functionalities. Software components are encapsulated into intelligent structures called agents as described in Chapter 5. The collection of all these agents working together is called a Multi-Agent System.

Multi-Agent Systems can adopt different organization structures depending on their purpose. Ferber and Gutknecht (1998) describe organizational MAS structures as a function of their role, interaction pattern, and communication language. For this research scope, three interaction patterns are considered: hierarchical, team, and federated. These patters are the most convenient to describe the onboard software of a satellite.

In the hierarchical organization pattern, a subset of agents interpreting a role define a structure in which an agent adopts the responsibility of coordinating the execution a task (coordinator), that is decomposed in pieces executed by specialized functional agents. The functional agents are subordinated to the coordination agent within the organization. A functional agents can not interact with other functional agents in the same organizational level. Instead, they interact with other agents outside the organization still within the system to achieve their local goal.

The team organizational pattern is similar to the hierarchical pattern in their structural configuration. The only difference is that there are not restrictions to functional agents within the same role to interact among them to achieve their local goal behavior or function. That allows more flexibility and reconfiguration capabilities in case of fault detection, which makes the subset of agents more resilient to failure.

In the federated approach, local organizations use either the hierarchical or team

structure to achieve a global goal. The idea of using a federated interaction strategy is to balance the communication load and the number of connections, which is convenient when there are communication constraints for the design.

Communication design pattern is a concept used describe the way design influences the abstraction of the communication process. They are often used to create a framework that ensures consistency and uniformity during the implementation phase. Communication design patterns can be used to minimize the implementation complexity of organizational structures in distributed systems. For instance, Hayden et al. (1999) describes communication patterns for MAS such as Broker, Embassy, Mediator, among others. There are two main advantages of using communication design patterns. First, the implementation complexity is reduced since the communication process can be generalized to be later tailored according the application's need. That increases the code re-usability, and reduces the implementation schedule. Second, the use of standardized communication improves the determinism of the system, which is critical in real-time systems, such as the ADCS. The establishment of a robust structure for agent's topology in a MAS-based software plays a significant role in their ability to achieve the consensus required in distributed processing. The next Section provides a summary of fault-tolerant consensus algorithms that can be implemented with MAS-based approaches.

6.2. MULTI-AGENTS SYSTEM CONSENSUS

Establishing consensus in parallel data processing is a big challenge in distributed computing systems. Software performance is profoundly influenced by the interaction topology employed to exchange information between computing agents. It is also influenced by external mechanisms such as delays and communication unbalance. Examples of these effects are presented and discussed by Tian and Liu (2009). Additionally, the speed of consensus is affected by the number of computing agents within the execution platform. Jin and Murray (2006) propose adopting multi-hop relay protocols to expand the knowledge of agents subsets and speed up its state convergence, which might benefit the system's performance. Consensus algorithms are used in multiple applications. They can be found in distributed state estimation with Kalman Filters as described by Ma et al. (2017) and distributed control by Wen et al. (2013). For space applications, Ren (2010) has demonstrated consensus algorithms for distributed cooperative attitude synchronization and tracking of rigid bodies. Also, in the space domain, MAS can be used for formation flying and fractionated spacecraft for long-term earth observation missions as discussed by Chu et al. (2013).

Software architectures using an agent-based FDIR architecture have been proposed in Chapter 3 to deal with the specific fault mechanism in sensors (e.g. drift). These architectures assume that fault detection signals are distributed among all the sensors, and it requires a few centralized agents to apply recovery algorithms to mitigate the impact of faulty measurements in the estimation of spacecraft's attitude. That is an excellent example of how consensus protocols are used in the design of reliable OBSW.

6.2.1. CONSENSUS STRATEGIES AND ALGORITHMS

There are two different types of consensus problems present when working with multi-agent systems. These are synchronous and asynchronous. The best-case scenario is when all agents are synchronized within a known time bound. The worst-case scenario is when systems operate asynchronously. For this Chapter, the focus is put in the asynchronous consensus algorithms.

According to Fischer et al. (1985), it is impossible to analytically solve consensus in asynchronous systems with one faulty process (e.g. Agent). Luckily, there exist alternatives such as the Paxos Algorithm described by Lamport et al. (2001) that provides safety and eventual liveness for reaching consensus within a time bound. Safety refers to the fact that only one of the proposed values is chosen and eventual liveness means that if there are not failures shortly, there is a good chance that consensus will be reached. Paxos also works in rounds, but the synchronization of these rounds is not required. If an agent is in its round and gets a new message or a timeout signal, it will abort what it currently doing and it move forward to round $r+1$. A ballot ID uniquely identifies each round, and it is broken down into two phases: 1) Election phase (Prepare/Promise), and 2) Bill (Accept/Commit) phase.

The implementation of consensus algorithms (e.g. Paxos) reduces the complexity of MAS-based software since it enables reusing already in-place capabilities such as the agent interaction protocols for communication, as well as agent management capabilities for failure recovery. Also, it balances the performance and reliability of the system to enable instant failure recovery with minimum overhead on computing cost.

6.3. TOPOLOGICAL OPTIMIZATION OF MAS-BASED SOFTWARE

This Section describes the optimization problem for agents communication using the organizational aspects described in Section 6.1. Also, it provides an overview of the search strategies considered in the solution of the optimal communication cost, particularly in the use of randomized optimization methods such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO).

6.3.1. TOPOLOGICAL MODELING OF MULTI AGENT-BASED SOFTWARE

A MAS-based software architecture can be described using the graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{I})$ in which $\boldsymbol{V}$ represents the nodes (agents), $\boldsymbol{E}$ corresponds to the communication links (edges) between these nodes, and $\boldsymbol{I}$ describes fixed interactions of agents connected through the communication links of the system. If there are n agents in the network, the interaction topology vector $\boldsymbol{x}$ represents the connection's state of the nodes in the graph $\boldsymbol{G}$. The probability of interaction $\boldsymbol{I}(\boldsymbol{T}_g, \boldsymbol{x})$ for a group of agents $\boldsymbol{T}_g$ that work together inside the system's graph $\boldsymbol{G}$ is assumed to be known during the software design process. A fixed interaction means that there is a interface constraint between specific agents within the software architecture. These fixed interactions are requirement driven.

In (6.1), the adjacency matrix $\boldsymbol{A}$ describes the connection of the nodes in graph $\boldsymbol{G}$, which is determined by the value of $\boldsymbol{x}$. In that expression, $\boldsymbol{A}_{ij} \in \{0,1\}$ meaning that $\boldsymbol{A}_{ij} = 1$ when there is direct an interaction between agents i and j and $\boldsymbol{A}_{ij} = 0$, otherwise. The expression in (6.1) was adapted from the work presented by Liu and Iwamura (2000).

$$\boldsymbol{x} = \{\boldsymbol{A}_{ij} : 1 \leq i \leq n-1, i+1 \leq j \leq n\} \tag{6.1}$$

This probability of interaction $\boldsymbol{I}(\boldsymbol{T}_g, \boldsymbol{x})$ is given the capability of message exchange between the nodes in the sub-organization $\boldsymbol{T}_g$. The total cost of interaction C_T for a topology candidate $\boldsymbol{x}^*$ can be quantified by the sum of the cost of each agent to reach out to other agents in the graph $\boldsymbol{G}$. The overall cost function for MAS-based software interaction given $\boldsymbol{x}^*$ can be calculated as

$$C_T = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \boldsymbol{c}_{ij} \boldsymbol{A}_{ij}(\boldsymbol{x}^*) \tag{6.2}$$

where, $\boldsymbol{c}_{ij}$ represents the weighted cost matrix for message exchange between agent i and agent j of the multi-agent based software architecture. The weighted cost matrix can be obtained experimentally by profiling the application and making an average of the required energy to transmit a message from agent i to agent j in the OBC. The matrix $\boldsymbol{c}_{ij}$ depends on implementation parameters (again driven by the mission requirements), which demands characterization before starting the process of topological optimization of the multi-agent based software architecture.

The fixed interaction constraints are determined by the specific organization within the MAS-based software architecture. For example, management capabilities require fixed functionality agents to be implemented (e.g AMS). On the other hand, there are groups of agents free to decide if they partner, or not. These are called flexible interaction constraints. The collective dynamics of these groups are described using the "Small Worlds" theory presented by Watts and Strogatz (1998) to interpolate the software architecture design between a regular graph and a random graph.

Now, consider the solution of the optimization problem formulated in (6.3) as an approach for the organizational optimization of n agents in a MAS-based software architecture with symmetrical communication cost. This description assumes a topology candidate $\boldsymbol{x}^*$ that contains l sub-organizations of agents, in which C_l represents the nodes that implement coordination roles for the l^{th} sub-organization of agents. These l sub-organizations are intended to implement functional tasks within the satellite software.

These sub-organizations can follow either a hierarchical or a team structure, and they are allowed a bigger degree of interaction with other nodes (agents) of the system.Fixed interaction constraints have assigned a higher priority to their communication with respect to other agents and teams dedicated to non-critical tasks.

Three primary roles are defined for the agents within the software architecture: (1) management, (2) coordination and (3) functional. Management agents (e.g. AMS) are dedicated to provide support and controlling the systems operations, while the coordination agents control the sub-organization of agents performing local-level tasks. Functional agents are specialized for concrete calculation tasks that demand their focus and performance. The optimization problem can be formulated as

$$min\, C_T = min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \boldsymbol{c}_{ij} \boldsymbol{A}_{ij}(\boldsymbol{x}^*) \tag{6.3}$$

subject to:
1) $\boldsymbol{A}_{1k} = 1; \quad k = 2,3,...,n;$
2) $\boldsymbol{A}_{cs} = 1; \quad c = 2,3,...,(n-1); s = 3,4,...,n;$
3) $deg(\boldsymbol{G}(C_l)) \leq \frac{n-1}{2} + 1; \quad l = 2,3,...,n;$
4) $deg(\boldsymbol{G}(F_m)) \leq 3 + round(n/10); \quad m = 2,3,...,n;$
5) $\boldsymbol{A}_{ij} = 0 \quad or \quad 1 \quad \forall \quad i,j$

In the optimization problem formulation proposed above, the first constraint is used to represent the fixed interactions for management activities between the Agent Management System and all the others agents in the MAS-based software architecture. The second constraint is intended to describe fixed interactions for both hierarchical and team organizations within architecture design.

The metric used controlling the flexible interactions constrains in the system is the degree function of a graph $deg(\boldsymbol{G}(i))$, which considers the connection of the node i in the graph $\boldsymbol{G}$ with the rest of the nodes. In the case of the AMS node, it can communicate to all the nodes in that graph. For the coordination and functional nodes, a limit in the number of interactions was assumed arbitrarily (mission dependent) so that they can scale appropriately with the number of agents in the network for any particular mission design. It is imperative to reach a balance between the clustering coefficient and the path length to achieve a small world behavior as described by Watts and Strogatz (1998). This is important to achieve a resilience level that takes advantage of distributed capabilities of MAS-based software.

BASIC MODELING EXAMPLE

This easy-to-understand example shows the application of the modeling concepts introduced above to present an analytical case study with a small network of agents. It also shows that once the size of the network increases, the search for a feasible solution for topology becomes difficult by hand, so automatic search algorithms are proposed and thus preferred.

Suppose that a MAS-based software system consists of 5 agents that control the operation of a system. The cost of communication for individual agents is one power unit among them. That implies that $\boldsymbol{c}_{ij} = 1 \quad \forall \quad i,j$ for simplicity. There is a constraint that imposes a hierarchical organization for agents $H = \{2,3,5\}$, where agent 2 is the coordinator and agents 3 and 5 are functional agents. According to the problem formulation in (6.3) the following constraints are defined:

1. Agent 1 is the manager (AMS) and has to communicate with all the agents in the system.

2. Agents 2,3 and 5 establish a hierarchy structure where agent 2 is the coordinator.

3. The degree of node 1 (AMS) of the graph must be 4.

4. The degree of node 2 (coordinator) has to be lower or equal to 3

5. The degree of all other functional nodes has to be lower than or equal to 3

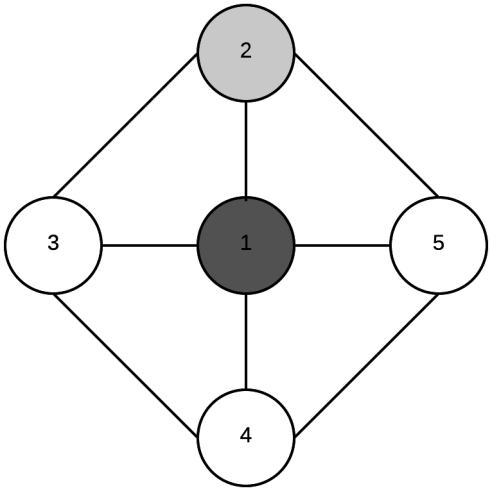


Figure 6.1: Basic MAS-based Software Architecture - Analytical Solution.

Based on these constraints, the optimal total cost was determined as $C_T = 8$ power units. This can be easily found analytically, and the optimal interaction topology is visualized in Figure 6.1.

The following adjacency matrix represents the optimal topology for the network, given that the optimal topology vector $\boldsymbol{x}$ was identified solving the optimization problem analytically as $\boldsymbol{x} = [1\,1\,1\,1\,1\,0\,1\,1\,0\,1]$:

$$\boldsymbol{A}(\boldsymbol{x}) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

6.3.2. NETWORK SCALE EFFECTS

In practical cases, the number of software functions in a MAS-based software architecture varies from 10-100. These functionalities require thousands of lines of code to get implemented as descibed by Dvorak and Lyu (2009). When the scale of the network increases, the overall cost optimization is no longer viable using analytical methods. Instead, computer-based algorithms are implemented to search for an optimal solution using in some cases the Brute Force Algorithm (BFA), which tries one by one all the possible solutions for find the one that fits the constraints. It has to explore the entire solution space in that search, which is not convenient.

Figure 6.2 shows the scale effects on the time to find a solution using the BFA search approach. It assumes that network sizes vary from 8 to 12 agents, and there are the fol-

lowing fixed interaction constraints: Hierarchy 1 (H1) composed of agents $H1 = \{3,4,5\}$, Hierarchy 2 (H2) composed of agents $H2 = \{6,7,8\}$, Team 1 (T1) $T1 = \{2,3,6\}$. It assumes that the cost of communication among each pair of agents is one unit, so that again $\boldsymbol{c}_{ij} = 1 \quad \forall \quad i,j$. Flexible interaction constraints are defined as a function of the number of agents in the network according to (6.3).

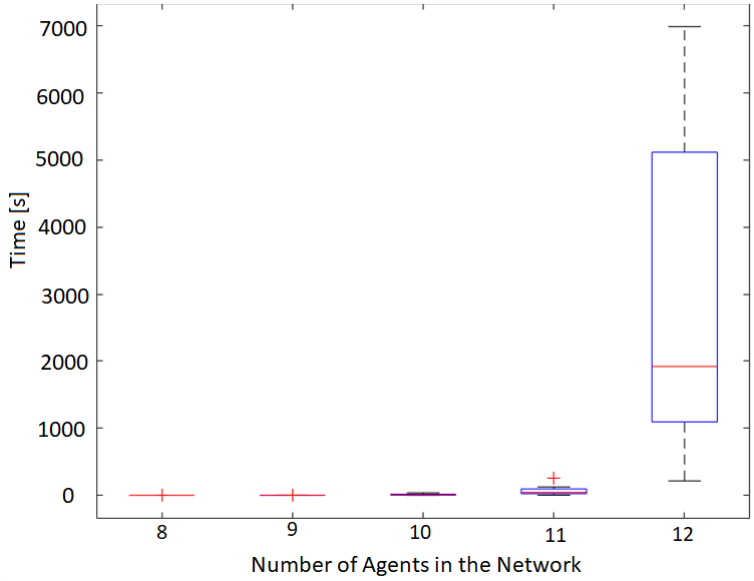


Figure 6.2: Network Scale Effect in the time for finding a solution using the Brute Force Algorithm implementation.

From Figure 6.2, it is explicit that as long as the network scales up the average time to find a solution increases very fast, as well as its variance. That situation motivates looking for a more effective search strategy, since using the Brute Force Algorithm search heuristic may lead to a very large (infinite) time for finding a feasible solution, even with less than 20 agents. During the simulation experiments with the BFA a problem size of 15 agents had to be stopped since it never found the optimal solution.

6.3.3. RANDOMIZED SEARCH STRATEGIES

Search heuristics are enablers for the solution of optimization problems. According to Auger and Doerr (2011) evolutionary algorithm such as genetic algorithms, simulated annealing, ant colonies and particle swarm optimization provide the performance required to solve big search problems in a reasonable time. In this Chapter, two families of search heuristics with evolutionary algorithms are explored: Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). Both algorithms are analyzed and compared to identify the most suitable for its implementation in a case study. These algorithms were selected since they provide a good performance to implementation complexity ratio.

GENETIC ALGORITHMS

Genetic Algorithms are an iterative population-based searching method that emulates the natural selection process. In this process, the GA frequently modifies a population of individual solutions that are called generations, according to mutation and crossover parameters that allow exploring the solution domain.

The workflow for implementation of GA described by Davis (1991) consists of three main steps. Initially, a selection of parents is made based on their feasibility to satisfy the problem constraints. This population of parents is then used in the second step for crossover, where two parents generate children for the next generation. Finally, mutation is implemented for applying random changes to generated children.

PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization is also an iterative population-based searching technique that explores a solution space emulating the behavior of movement of organisms. Shi and Eberhart (2001) describes how PSO can be adapted on four different levels (environment, population, individual, component) to achieve faster convergence and better performance. This is convenient when there are computational limitations in the optimization process.

The workflow in the PSO algorithm is as follows. First, a set of potential solutions (particles) are initialized. Then the fitness of these solutions is calculated and compared to the personal best value; then it is updated. Otherwise, the current personal value is kept. After that, each particle compares its fitness value with the global best value, the better performing particle is then assigned to the global best value and the internal state of each particle is updated for the next iteration. This cycle repeats until error threshold for the search are achieved.

PSO VS GA

Several comparisons between PSO vs. GA can be found in the literature. For instance, in Hassan et al. (2004), Panda and Padhy (2008) and Roberge et al. (2013) several aspects of PSO and GA are assessed and compared, such as computational requirements, convergence speed and performance limitations (local vs global optimum).

One of the biggest differences between GA and PSO is the mechanism used to generate a population of solutions. They have different strategies to balance between exploration and exploitation, which can affect the converging time. Kachitvichyanukul (2012) argues that Genetic Algorithms have less tendency for premature convergence, which is desirable for achieving global optimum, compared to PSO and differential evolutionary algorithms.

6.4. OPTIMIZATION IMPLEMENTATION

Genetic Algorithms were chosen for solving the optimization due to their ability to balance exploration and exploitation capabilities in achieving a feasible solution that satisfies the constrains of communication budgets for MAS-based software. The exploration-exploitation trade-off is a fundamental dilemma whenever you need to learn about the solution domain by trying things out new combinations. The dilemma is between choosing what you know with a current solution and getting closer to what you expect ('exploitation'), and choosing something new you aren't sure about and possibly learning more ('exploration'). This trade off has a direct impact on the time required to complete the search for an optimal solution. Also, the implementation complexity was found to be a determining factor to decide for GA instead of PSO. This section describes the way the GA was coded using $Python$.

The topology interaction candidates for a MAS-based software architectures are represented using a row vector $\boldsymbol{x}^*$ with dimensions $1 \times [n(n-1)/2]$ as in (6.1). Vector $\boldsymbol{x}^*$ is considered as a "chromosome" candidate composed of "genes" describing the interaction between agents (nodes) in the graph. Inside the chromosome, there are fixed genes (bits) representing hardwired interaction constrains in the candidate topology. These genes are called "golden genes" and they cannot be mutated in later generations since they represent the fixed interactions defined by design.

The initialization process for the genes of a chromosome is implemented considering the fixed interaction constraints. The genes representing the organization interactions (hierarchical or team) are set to '1', while the rest of them are randomly assigned to either 0 or 1. That procedure guarantees the feasibility of a chromosome. The process is repeated until the algorithm generates the number of chromosomes specified by the target population parameter pop_{size}.

The evaluation of chromosomes determines their fitness for further reproduction and genes propagation. In the proposed algorithm the probability of reproduction is determined based on a fitness function that determines how a candidate solution $\boldsymbol{x}^*{}_i$ satisfies the flexible interaction constraints (number 3 and 4) in (6.3). The expression in (6.4) shows that the fitness value depends on the number of the interaction between coordination and functional agents. The fitness value is normalized for further processing during the candidate solution selection. F_j is a subset of agents in candidate architecture $\mathbf{x}^*$ representing the functional agents of that architecture that are free to interact with other agents to achieve their goals.

$$Fitness(\boldsymbol{x}^*) = \begin{cases} 0 & \text{Flexible interactions are not satisfied,} \\ \frac{\sum_{j=1}^{s} F_j(\boldsymbol{x}^*)}{s} & \text{otherwise.} \end{cases} \tag{6.4}$$

There are several selection schemes for evolutionary algorithms as discussed in the work presented by Blickle and Thiele (1996). For the proposed algorithm in this Chapter the fitness value is calculated using the expression above. After that, chromosomes are organized from higher to lower fitness score for the topology selection process. This is done this way to ensure better candidates propagate their features first, like in natural selection.

The proposed algorithm uses the approach proposed by Miller et al. (1995), which consists in ordering the chromosomes by fitness value and choosing with higher probability the ones with better fitness for crossover and mutation. This operation requires defining a parameter for establishing the size of the tournament selection T_{SS}.

The crossover operation also requires defining a parameter (P_C) for describing the probability of this operation to happen. This parameter is used for tuning the speed of convergence of the algorithm. The genetic algorithm selects, pairs and evolves chromosomes to increase the fitness of the elite chromosomes in the population. This process is repeated until the stop condition that satisfies the architectural design requirements is reached. For the case study, a convergence criteria will be defined so that it can be implemented and compared for all algorithms.

Figure 6.3 depicts the flow diagram that was implemented for the simulation. In that diagram, the size of the problem, the constraints interaction and the genetic algorithm optimization (GAO) parameters are input to describe the simulation scenario. These are taken as reference for the execution and data collection according the algorithm described above.

The implemented algorithm also offers features to visualize the most fitted topology configuration, as well as its fitness and total communication cost evolution over time. The next Section illustrates the applicability of the proposed GA optimization for satellite subsystems.

6.5. TOPOLOGICAL OPTIMIZATION FOR AOCS SOFTWARE

This Section illustrates the application of the proposed GA for the optimization of the total communication cost of onboard software components for hypothetical operation scenarios of a real satellite mission. For that purpose, the PROBA 3 mission developed by ESA was selected, due to its performance and robustness requirements for the AOCS subsystem in particular.

6.5.1. PROBA 3 MISSION DESCRIPTION

According to Llorente et al. (2013), the PROBA 3 mission is aimed towards increasing the confidence and TRL of formation flying technologies. It will demonstrate metrology sensors for formations flying, advanced formation control algorithms, and onboard FDIR capabilities. The space segment of PROBA 3 is composed of two independent three-axis stabilized mini-satellites in highly-elliptical Earth orbit, performing precise formation flying, so that they are able of controlling the attitude and separation of both satellites accurately.

The concept of operations can be summarized as follows. When both satellites are separated from the launcher, they will remain together in a stacked configuration, so that the propulsion system installed on the bottom satellite is able to move them to the nominal operation orbit. Once they reach the target orbit, they will separate and change to a safe relative trajectory that avoids potential collisions. During the commissioning phase, several subsystems and software components will be tested and validated until completely autonomous operations are reached. Finally, the formation flying exercises will begin to test formation acquisition, re-configurations, manoeuvres, accuracy, and

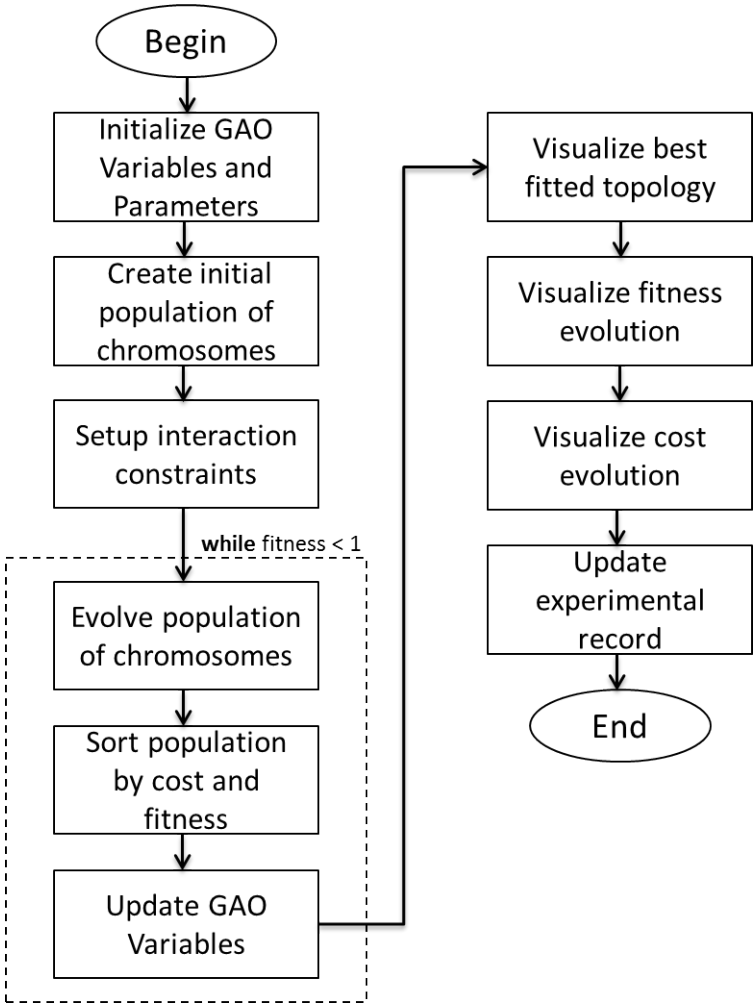


Figure 6.3: Genetic algorithm optimization flow diagram for organizing MAS-based software architectures.

commands exchange between both satellites. At the end of life the satellites will be de-orbit to satisfy international regulations on space debris.

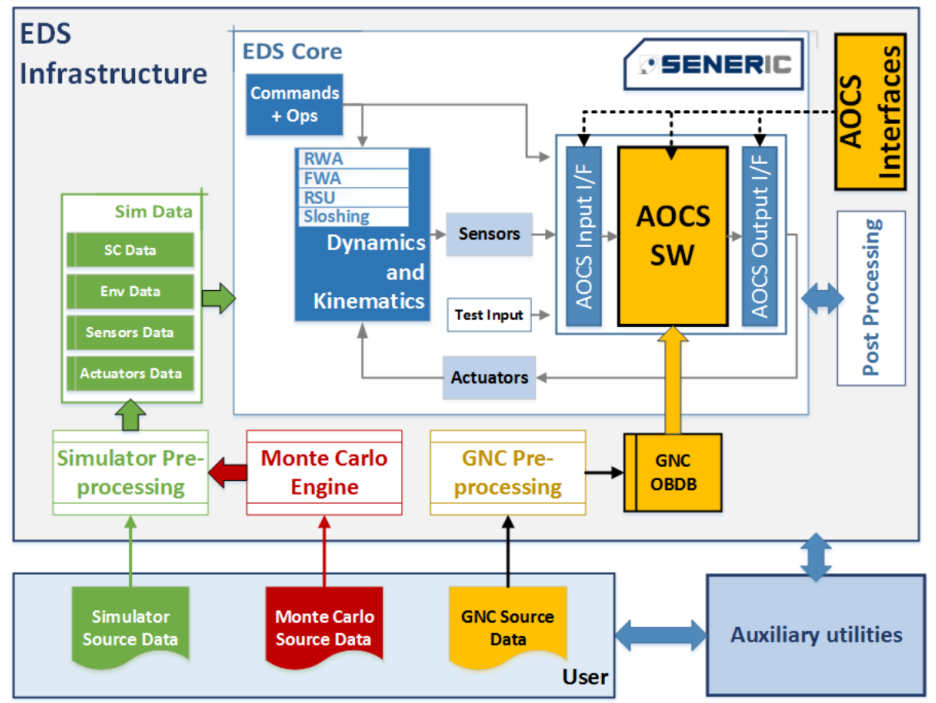
The AOCS for PROBA 3 is very sophisticated. It incorporates 6 degrees of freedom formation control with thrusters, collision avoidance sensors and rendezvous capabilities. In the work of Borde et al. (2004), different sensors used in PROBA 3 for precise formation flying are described. These include coarse and fine radio-frequency sensors, star tracker, coarse sun sensors, gyroscopes, coarse and fine optical lateral metrology, fine optical longitudinal metrology and GPS. The installed sensors will enable formation precision in a range down to milli-arcsecond. As actuators, the leading satellite will have eight thrusters and six tanks to perform control experiments. Also, it includes three reaction wheels and magneto-torquers.

Regarding the software, the onboard computer will execute the navigation and guidance function, as well as, the control commands for the relative position and relative attitude of the both spacecraft. It will require multiple software components (agents) cooperating to achieve the formation flying goals for each experiment, while satisfying the operation budgets. For the AOCS implementation the following modes were defined: sun acquisition mode, coarse transition mode, normal mode, and orbit control mode. Also there is a stand-by mode, and FDIR modes for fault recovery.

For the simulation scenarios, the AOCS modes will be simplified and assumptions will be made to make them more illustrative in terms of the proposed GA proposed to organize the onboard software agents. The details of the simulation scenarios are explained in the following Subsection.

6.5.2. SIMULATION SCENARIOS

The simulation scenarios were built to satisfy the AOCS software architecture for PROBA 3 shown in Figure 6.4 as described by Cacciatore et al. (2016)



EDS = Euclid Design Simulator
GNC = Guidance Navigation and Control
AOCS = Attitude and Orbit Control System
SW = Software
SC = Spacecraft

RWA =Reaction Wheel Assembly
FWA = Filter Wheel Assembly
RSU = Remote Sensor Unit
OBDB = Onboard Data Base
Ops = Operations

Figure 6.4: Software Architecture for the AOCS of PROBA 3 Mission by Cacciatore et al. (2016)

The implementation of the AOCS modes for the case study about PROBA 3 followed the diagram in Figure 6.5, which shows the required inputs and output for each AOCS mode.

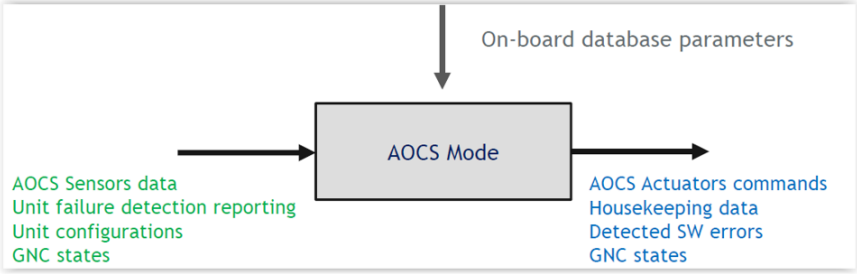


Figure 6.5: Information flow for the implementation of PROBA 3 AOCS modes by Cacciatore et al. (2016)

Three simulation scenarios were synthesized in order to test the scalability of the proposed Genetic Algorithms implemented for organizational optimization. These are called the (a) Safe , (b) Normal and (c) FDIR scenarios. The Safe scenario describes the operation during the Sun acquisition mode for the AOCS. The Normal scenario describes the nominal operation mode of the AOCS, and the FDIR scenario describes the addition of FDIR features to the normal mode to make it more robust. The number of software agents and the interaction constraints are defined based on mission basis, so the performance of the proposed algorithm can be assessed with respect to the increase in the number of software agents. The main idea is scaling up the number of agents between the scenarios to assess how the performance of the proposed GA is affected by the size of the problem.

SAFE SCENARIO FOR AOCS

This scenario consists of 10 software agents interacting together inside the AOCSOBC to perform the Sun acquisition maneuver described by Borde et al. (2004) , which is also used as the safe mode for attitude control. The agent's functionality are described as follows:

- Agent 1, called AMS monitors all the other agents to collect performance data.
- Agent 2, called CAL is assumed to perform calibration procedures to sensors and actuator on the system based on operations request.
- Agent 3 called ParDB is in charge of controlling the interaction with the on-board database for operations parameters.
- Agents 4 and 5, called OrbEst, and AttEst, respectively, are in charge of the state estimation of the spacecraft.
- Agents 6 and 7, called OrbCont, and AttCont are used to execute the orbit and attitude control algorithms, respectively.

- Agent 8 called GyroMan implements the software capabilities required for managing the 3-axis gyroscope.
- Agent 9 called SSMan implements the software capabilities required for managing the 3-axis Sun sensors.
- Agent 10 called PropMan implements the software capabilities required for managing the 8 thrusters of the system.

Two hierarchical interactions must be satisfied during this mode. Firstly, once a command is received, the agent 2 (CAL) shall subordinate agents 8, 9 and 10 for them to execute the calibration procedure. Second, when a change in the operation parameters of the estimators is required, the agent 3 (ParDB) will subordinate 4 and 5 to their update. Also, two team interactions are defined for this scenario. One is related to the attitude estimation, where agent 4 (AttEst) team up with agents 8 and 9 for current spacecraft attitude estimation. The second team is for agent 5 (AttCont) interacting with agent 10 to control the orientation of the satellite to keep pointing to the Sun. The AMS must be able to interact with all agents in the organization at any moment.

NORMAL OPERATION SCENARIO

The normal operation scenario requires 20 agents to describe how the spacecraft behaves under nominal conditions. According to Borde et al. (2004), during the Normal Mode of PROBA 3 its AOCS shall control the attitude of the spacecraft for its own purpose and for performing the formation flying experiments described above. That includes Sun pointing maneuvers with Reaction Wheels and Star Tracker, controlling the attitude during the experiment, specifically estimating and controlling the distance between satellites, using various sensor such as R-GPS, RF or optical metrology. Table 6.1 summarizes the list of required agents for the implementation of this operation scenario.

The following interaction constraints shall be implemented during the normal operation scenario:

- The AMS agent shall be able to interact with all agents in the organization at any moment, forming a hierarchical structure.
- The calibration agent (CAL) interacts with agents 8, 9, 10, 11, 12, 13, 14, 15 forming a hierarchy.
- The parameters database agent 3 (ParDB) interacts with agents 4, 5, 18, and 19 forming a hierarchical structure.
- The attitude estimation agent 4 (AttEst) forms a team structure with agents 8, 9, 11 and 12 for determining the state of its orientation.
- Finally, a team structure was created between the attitude control agent 5 (AttCont) and agents 10, 12 and 17 for the execution of the attitude control algorithm.

FDIR OPERATION SCENARIO

This mission scenario extends the nominal mode introduced in the previous Section with FDIR features to provide a more robust performance. It consist of 30 agents including the agents in Table 6.1 and the listed agents in Table 6.2 to provide FIR capabilities to the AOCS software.

It is important to remember that based of the FDIR architecture proposed in Chapter 3 the FDI capabilities are integrated on agents behaviors for sensors and actuators, and the FIR capabilities for each of them are implemented separately as described in Table 6.2.

With respect to the constraints in interaction, FIR agents are free to interact with any of the agents in the organization to achieve their goals. The motivation behind letting them be free is for experimental consistency purposes, meaning that in order to keep the constant the fixed communication cost among all the operation scenarios enable a better comparison of the performance of the implemented GA. The next Section focuses on how the simulation setup was configured.

Table 6.1: Functional description of agents required for the normal operation scenario

Agent ID	Agent Name	Agent Function
1	AMS	This agent is in charge of controlling the lifecycle of all the agents within the organization. Also, it collects performance data.
2	CAL	The calibration agent works under request from the operators of the mission. It perform calibration procedures to sensors and actuator on the system.
3	ParDB	ParDB is in charge of controlling the interaction with the on-board database for operations parameters.
4	AttEst	This agent implements the estimation algorithm for the attitude of the spacecraft.
5	OrbEst	This agent implements the estimation algorithm for the orbit of the spacecraft.
6	AttCont	This agent implements the control algorithm designed to keep the attitude during the formation flying experiments.
7	OrbCont	This agent implements the control algorithm designed to maintain the orbit during the formation flying experiments.
8	GyroMan	GyroMan implements the software capabilities required for managing the 3-axis gyroscope.
9	SSMan	SSMan implements the software capabilities required for managing the 3-axis Sun sensors only during the Safe Mode.
10	PropMan	This agent is intended to provide the capabilities required operating the 8 propellant thrusters onboard the spacecraft.
11	STMan	This agent controls the operation of the Star Tracker for the Sun Pointing manuever
12	RWMan	RWMan manages the operation of the Reaction Wheels onboard the spacecraft.
13	MTQMan	There are 2 magneto-torquers onboard for off-loading the reactions wheels during the Sun pointing. This agent controls their operation.
14	RGPS	This agent controls the R-GPS sensor at the beginning of the mission when it is used as sensor input to control the inter-satellite distance.
15	RFMet	RFMet agent provides the software capabilities for performing radio-frequency metrology during the formation flying experiment.
16	OptMet	This agent provides the software capabilities for performing optical metrology during the formation flying experiment.
17	CGTrust	CGTrust is an agent to control the cold gas thrusters to keep the inter-satellite distance.
18	ISDEst	This agent uses either the R-GPS, RF or optical metrology to estimate the inter-satellite distance during the formation flying experiment.
19	ISDCont	This agent implements the control algorithm to keep the inter-satellite distance according to specifications during the formation flying experiments.
20	FFTMT	FFTMT collects telemetry related to the formation flying experiments so that it can be downloaded and analyzed later.

Table 6.2: Arbitrary defined FIR agents for supporting the nominal operations of PROBA's 3 AOCS

Agent ID	Agent Name	Agent Function
21	FIR_Gyro	This agent enables gyroscopes identifying, isolating and establishing a recovery procedure for faults.
22	FIR_SS	This agent enables Sun sensors identifying, isolating and establishing a recovery procedure for faults.
23	FIR_ST	This agent enables star trackers identifying, isolating and establishing a recovery procedure for faults.
24	FIR_RF	This agent enables the RF metrology identifying, isolating and establishing a recovery procedure for faults.
25	FIR_Opt	This agent enables the optical metrology identifying, isolating and establishing a recovery procedure for faults.
26	FIR_CGT	This agent enables cold gas thrusters identifying, isolating and establishing a recovery procedure for faults.
27	FIR_GPS	This agent enables the R-GPS identifying, isolating and establishing a recovery procedure for faults.
28	FIR_MTQ	This agent enable magneto-torquers identifying, isolating and establishing a recovery procedure for faults.
29	FIR_RW	This agent enables the reaction wheels metrology identifying, isolating and establishing a recovery procedure for faults.
30	FIR_MEM	This agent enables the onboard memory identifying, isolating and establishing a recovery procedure for faults.

6.5.3. SIMULATION APPROACH

After the simulation scenarios were introduced, it is necessary to describe the setup required to collect performance data for the proposed GA algorithm implemented to minimize the total cost of communication within the MAS organization. The input variables and parameters used for that implementation are summarized in Table 6.3.

Table 6.3: Description of inputs considered during the implementation of the proposed GA

Name	Description	Type of Input	Data Type
NUM_AGENTS	Number of agents in the software organization	Variable	Integer
COST_MAT	Relative cost matrix for inter-agent communication	Variable NxN Matrix	Floating-Point
HIERARCHIES	Hierarchical constraints that shall be satisfied during the solution of the optimization problem.	Variable List of Arrays	Integer
TEAMS	Team constraints that shall be satisfied during the solution of the optimization problem.	Variable List of Arrays	Integer
NUMB_OF_ELITE_CHROMOSOMES	Number of elite chromosomes for the Genetic Algorithm implementation.	Constant Parameter	Integer
TOURNAMENT_SELECTION_SIZE	Number of "tournaments" executed to select two best individuals among these k individuals to be parents.	Constant Parameter	Integer
MUTATION_RATE	This parameter is used to maintain genetic diversity from one generation to the next in the topology candidate population.	Constant Parameter	Floating-point between 0 and 1
POPULATION_SIZE	Number of topology candidates per generation	Constant Parameter	Integer

The operation of the GA is controlled by four variables and four parameters. The variables are the number of agents, the relative cost matrix, the hierarchies and the team constraints of the organization. For each case study scenario these variables need to be specified. Also, it is important to discuss how to select the proper values for the parameters of the GA. As shown in Table 6.3, four parameters determine the performance of the algorithm. These are the number of elite chromosomes, the size of the tournament selection pool, the mutation rate and the population size.

Elitism reserves some slots in the next generation for the highest scoring chromosome of the current generation, without allowing that chromosome to be crossed over in the next generation. The tournament selection size is used to choose k random individuals from the population and select the best two individuals among these k individuals to be parents in the next generation. If the tournament size is large, weak individuals have a smaller chance to be selected, which causes loss of diversity and therefore chances of getting stuck in a local optimum.

The mutation rate is used for controlling the exploration of the search space, more precisely to allow candidates to escape from local optima. In case of a large mutation rate, the population has difficulties to converge to a global minimum so that it needs to be balanced to achieve a solution within a reasonable number of iterations.

Concerning the population size, there is not a consensus about how to systematically select it. However, it is clear that it depends on the size of the problem, which is determined by number of agents in the architecture.

The implemented algorithm provides a set of outputs that enable obtaining the best topology candidate, as well as its cost, fitness, number iterations (generations) and execution time required to optimize communication cost. The output details are summarized in Table 6.4.

Table 6.4: Description of output parameters for the implemented GA

Name	Description	Data type of output
CANDIDATE_TOP	Is the most fitted topology produced by the genetic algorithm	1-D array of binary values
NUM_ITERATIONS	Number of iterations before achieving the target fitness	Integer Value
FITNESS	Measurement of how good is the solution given by the algorithm	Floating-Point between 0-1
TOTAL_COST	Is the total communication cost for candidate topology	Floating-point
EXEC_TIME	Time required to find a solution that satisfies the fitness target	Floating-point

Using the input variables synthesized in Table 6.3, the simulation design layout in Table 6.5 was implemented and executed.

Table 6.5: Design of Experiment for characterizing the performance of the proposed GA

NUM_AGENTS	HIERARCHIES	TEAMS
10	Minimal	Minimal
	Nominal	Nominal
20	Minimal	Minimal
	Nominal	Nominal
30	Minimal	Minimal
	Nominal	Nominal

The number of agents in the simulation implementation varies based on the operation scenarios described in Section 6.5.2. The fixed interaction constraints are defined as minimal, when the AMS is able to interact with all the agents, and both hierarchical and team structures are reduced to a single fixed interaction, while it is described as nominal when they correspond with ones defined in the description of the operations scenario.

For simplicity purposes the value of COST_MAT was normalized to one, so that cost of communication between agents is assumed to be homogeneous.

Regarding the simulation parameters, Table 6.6 summarizes the values used for this simulation scenario as a function of the problem size (NUM_AGENTS).

Table 6.6: Simulation parameters as a function of the number of agents in the software architecture.

Simulation Parameter	Value
NUMB_OF_ELITE_CHROMOSOMES	Round(NUM_AGENTS/4)
TOURNAMENT_SELECTION_SIZE	Round(NUM_AGENTS/2)
MUTATION_RATE	1/(NUM_AGENTS*(NUM_AGENTS-1)/2)
POPULATION_SIZE	NUM_AGENTS

The simulation was used for understanding the effect of the size (NUM_AGENTS) in the performance of the GA implemented for the optimization of the total cost of communication in MAS-based software architectures. The results obtained with this experimental configuration are presented and discussed in the next Section.

6.5.4. RESULTS AND ANALYSIS

This section presents the results for the implementation the GA for total communication cost optimization according to the outputs described in Table 6.4 and the design of experiment in Table 6.5 and the flow diagram in Figure 6.3. For that purpose, the simulation experiments described above were repeated 30 times to collect information with statistical significance. For each simulation scenario, it presents the best and worst result, as well as comparative figures for all operations scenarios.

SAFE OPERATION SCENARIO

The execution of the simulations was performed as described in Table 6.5 for 10 agents using the set of nominal interactions described above. For the Safe operation scenario, the best and worst result for the software architecture's topology is shown in Figure 6.6. From the results obtained for the Safe scenario, the algorithm took between 4 and 83 generations to find a feasible solution for minimizing the total cost of communication problem from (6.3) using the fitness function defined in (6.4). The solutions obtained varied between 20 and 24 power units, with an average total communication cost of 22 power units. In Subplots A and B, the fixed interactions are drawn as solid lines, whereas dotted lines are the one proposed by the GA to satisfy the flexible interaction constraints.

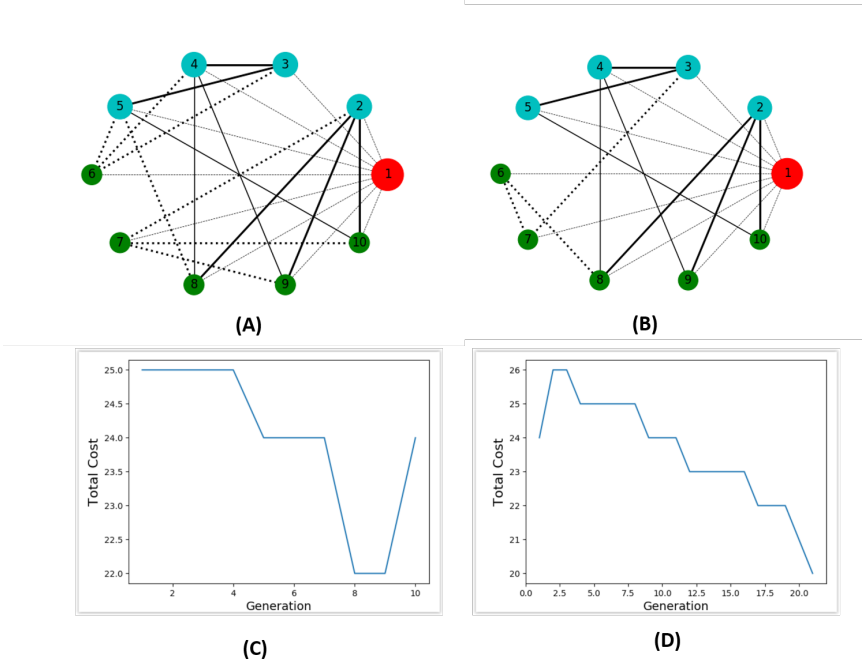


Figure 6.6: Results of the implementation of the proposed GA for optimizing the cost of communication in agent-based software architectures for the Safe Operation Scenario of PROBA 3 AOCS case study. (A) Worst MAS organization topology generated. (B) Best MAS organization topology obtained. (C) Total Communication Cost evolution for the worst result obtained with the GA. (D) Total Communication Cost evolution for the best result obtained with the GA

NORMAL OPERATION SCENARIO

The results of running the simulation 30 times for Normal operation scenario are depicted in Figure 6.7. Subplot A shows that the worst case solution obtained for the total communication cost was 61 cost units, compared to 49 cost units for the best solution in Subplot B. In both Subplots A and B the fixed interactions are shown in solid lines. The hierarchical constraints are shown in the thicker solid line, while the teams are drawn as thin solid lines. Agent 1 (AMS) interacts with all the agents in the organization. All coordination agents (2, 3, 4 and 5) are colored differently to show a different role within the organization. The rest of agents are drawn green to symbolize they are functional agents without any coordination responsibility.

The number of generations required to achieve a feasible solution in the Normal operation scenario varied from 54 to 1400. The average number of generations required was 195. The average total communication cost for this experiment was calculated as 54 cost units. The time to find a solution was in average 98 seconds. Subplots C and D show a particular behavior related to how the algorithm handled local vs. global optimum. In Subplot C around generation 30, the algorithm found a local optimum, so it increased the exploration to see if this was a local or a global optimum, then it found that required more iterations to find a globally optimal solution. The same behavior was observed in

Subplot D around the 80^{th} generation.

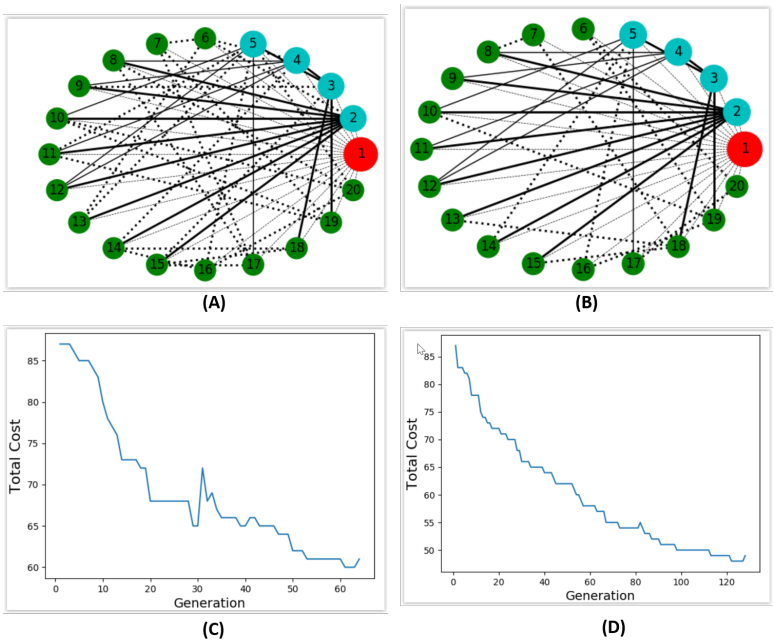


Figure 6.7: Results of the implementation of the proposed GA for optimizing the cost of communication in agent-based software architectures for the Normal Operation Scenario of PROBA 3 AOCS case study. (A) Worst MAS organization topology generated. (B) Best MAS organization topology obtained. (C) Total Communication Cost evolution for the worst result obtained with the GA. (D) Total Communication Cost evolution for the best result obtained with the GA

FDIR OPERATION SCENARIO

The FDIR operation scenario uses the same set of fixed interactions constraints as the Normal Operation Scenario. It gives freedom to the FIR agents to interact freely with all the others agents for isolation and recovery configuration. Same as before, Subplots A and B show the worst and the best solution obtained using the algorithm. Then Subplots C and D show how the total cost of communication evolves. The spikes in these evolution profiles show the ability of the algorithm to balance exploration and exploitation capabilities for achieving a globally optimal solution. It is clear that the longer the algorithm evolves, the better solution that is achieved. That is the same for all the operational scenarios.

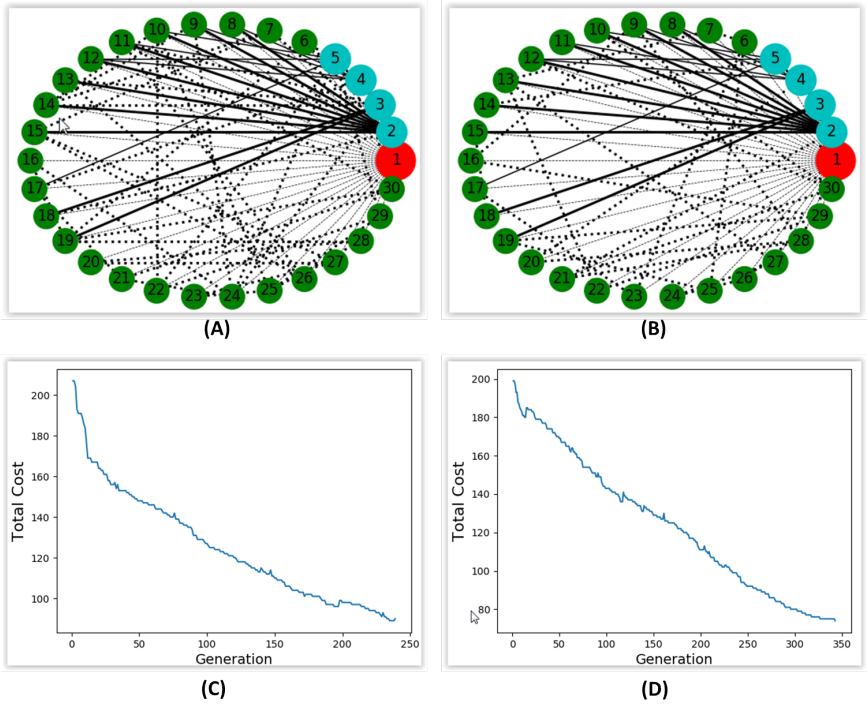


Figure 6.8: Results of the implementation of the proposed GA for optimizing the cost of communication in agent-based software architectures for the FDIR Operation Scenario of PROBA 3 AOCS case study. (A) Worst MAS organization topology generated. (B) Best MAS organization topology obtained. (C) Total Communication Cost evolution for the worst result obtained with the GA. (D) Total Communication Cost evolution for the best result obtained with the GA

ALGORITHM PERFORMANCE ANALYSIS

This part of the experiment consisted in setting the minimal number of interactions and running the proposed GA 30 times for 5 different scenarios with 10, 20, 30, 40 and 50 agents to collect performance data on the total communication cost, the number of generations and the time required to find a solution. Figure 6.9 shows how these output variables vary according to the number of agents required to implement a particular

MAS-based software architecture. For this test, the parameters of implemented GA were tuned to get the best result for simulation time. Table 6.7 shows how the parameters of the algorithm were set. These parameters were adjusted using a trial and error approach.

Table 6.7: Simulation parameters used to characterize the problem size effect in the optimization solution with the proposed GA

Simulation Parameter	Value
NUMB_OF_ELITE_CHROMOSOMES	4
TOURNAMENT_SELECTION_SIZE	8
MUTATION_RATE	1/(NUM_AGENTS*(NUM_AGENTS-1)/2)
POPULATION_SIZE	10

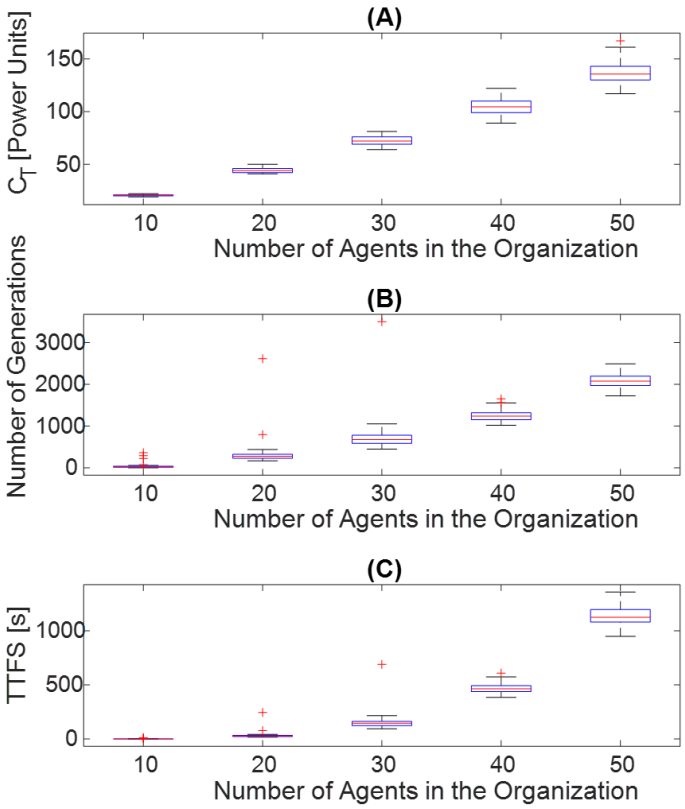


Figure 6.9: Comparison of (A) Total Communication Cost (C_T), (B) Number of Generations and (C) Time to find a Solution (TTFS) for 5 operations scenarios to assess the effect of increasing the problem size in the GA output's performance.

RESULTS DISCUSSION

After the execution of the experiments, the obtained results were analyzed for understanding the behavior of the output variables as a function of the size problem size. The second element analyzed was the effect of changing the GA parameters on the algorithm's performance. The third aspect analyzed was a variance of the obtained solutions as a function of the problem size, and finally, the GA's exploration vs. exploitation performance was observed to understand how the algorithm dealt with local optimal solutions.

As shown in Figure 6.9, either the total communication cost (C_T) as well as the number of generations required to find a possible solution scale linearly with the size of the problem. However, for the time required to find a solution, this is not the case. After post-processing, the data the best fit for a function able to predict the time of finding a solution as a function of the problem size can be expressed using a polynomial function of order 3. The coefficient of determination obtained for this case was $R^2 = 0.97$. When comparing the performance of the implemented GA to the BFA shown in Figure 6.2, it is clear that the first one is not only faster but also able to find solutions with less time variability, which is a desirable feature for a design tool.

The effect of changing the implementation parameters for the proposed GA was assessed using the three operation scenarios described above. The configurations of these parameters were set according to Table 6.6 and Table 6.9. For each operation scenario the total communication cost, number of generations and time to find a solution were analyzed using a 2-sample t-test to determine statistical significance for both sets of parameters. The analysis gave no statistical significance for the total communication cost in any of the scenarios. That means that in all cases, no matter the configuration the proposed GA can converge to a global solution. The two-sample t-test was performed assuming equal variance for the input data, so it made the test more strict.

The variance of the solutions obtained using the proposed GA was analyzed using the minimal fixed interactions with the tuning parameters from Table 6.9. It was observed that when the number of agents in the organization increased, the variability of the solutions obtained also increased. That has to do with the fact that the number of feasible solutions is determined by the number of agents as well so that there is a direct correlation between the solution variability and the size of the problem. This behavior can be observed in (6.3) where the flexible interactions are defined arbitrarily as a function of the problem size.

The final aspect to discuss after the implementation and characterization of the proposed GA is the balance between exploration and exploitation of the knowledge of the algorithm to reach a feasible solution as soon as possible. It was controlled with implementation parameters of the proposed GA. It was proven that reducing the population size, the tournament selection size and the number of elite chromosomes impacted the time of convergence of the algorithm positively, but increased the number of generations required to find a feasible solution. The mutation rate was kept constant as a function of the problem size to avoid falling in locally optimal solutions so that the algorithm was able to increase exploration over exploitation when the algorithm found a family of candidate solutions.

6.5.5. VALIDATION OF RESULTS

The validation of the results obtained from the development of this Chapter was divided into two parts. In one hand, there was a comparison of the results of the proposed GA with other search algorithms (e.g, BFA) to discuss the impact in the performance and the quality of the numerical results was performed. On the other hand, the applicability of this algorithm for satellite's AOCS software design had to be demonstrated.

Regarding the performance of the proposed GA, it was compared to a Brute Force Algorithm implemented to compare the results obtained for similar problem size and interaction constraints. Concerning the ability to find a solution, the BFA was able to find a solution for up to 13 agents, while the proposed GA was able to solve problems with 50 agents. The GA shown less time variability compared to the BFA. Both, the BFA and the GA were compared experimentally using the same programming language and problem sizes as shown in Figures 6.2 and Figure 6.9 (plot C), respectively. In terms of time performance, the proposed GA is up to 200 times faster than the BFA to find a feasible solution.

From the application point of view, the presented operation scenarios were compared with technical documents to validate the number and type of functionalities required to be implemented within the OBSW of a satellite mission. There was a good correspondence between the number of functions and the capabilities required so that the proposed scenarios were deemed realistic.

6.6. CONCLUSIONS AND REMARKS

This Chapter has introduced and described the use of Genetic Algorithms for organizational optimization of MAS-based software architectures. The algorithm was used to illustrate its applicability in a case study with a highly autonomous mission inspired on PROBA 3 AOCS operational modes.

This Chapter provides a mathematical formulation for the optimization problem of minimizing the communication cost in Multi-Agent Systems under fixed and flexible interaction constraints. It also proposes the adoption of randomized search strategies. The selection of the optimization algorithm included the assessment of Genetic Algorithms and Particle Swarm Optimization, resulting in the design and implementation of an easy to use tool for organizational optimization of MAS-based software architectures.

The Chapter provides a characterization of the proposed algorithm to show performance aspects such as time to find a solution and number of generations required to find a feasible solution so that designers can improve and speed up their software development process.

The algorithm shows that independently from its implementation parameters it is able to find a global solution that satisfies the proposed constraints. It also showed that keeping fixed the implementation parameters is beneficial for the performance of the algorithm from a time perspective.

Future work will consider the implementation of additional strategies such as PSO for comparison purposes. Also for the implemented GA, a sensitivity analysis will be conducted for identifying the best operating point of the algorithm for MAS-based software topological optimization. Also, as part of the improvements of the algorithm, the value of the parameter matrix COST_MAT needs to be estimated for each mission.

7

CONCLUSIONS AND OUTLOOK

"I believe in the future. It is wonderful because it stands on what has been achieved."

Sergei Korolev

Abstract

This Chapter summarizes the conclusions obtained as the result from the investigation formulated in the Introduction chapter. Also, the main innovations and contributions of this thesis are highlighted as well as the outlook for future research. Finally, some recommendations for this research are given for those working on similar topics or in similar fields.

7.1. RESEARCH SYNTHESIS AND CONCLUSIONS

This thesis focused on the development of a paradigm for designing and implementing agent-based onboard software architectures to support fault tolerance in the onboard software of small satellite subsystems. The proposed approach is able to handle the increasing onboard software complexity, and it enables native features for fault detection, isolation, and recovery of critical system components in small satellite missions.

The architectures, models, and methodologies presented on this dissertation can be used as tools for designing more robust software required for performing more complex space missions involving autonomous operations and high-speed ground link communications. During the development of this dissertation, three research questions (RQs) were defined in Section 1.7.2. The first research question was divided in two parts, one to the address the application's implications of agent-based software architectures and another to address the infrastructure required for implementing agent-based software architectures. The other two research questions (2 and 3) were addressed in Chapter 5 and Chapter 6, respectively. The following paragraphs elaborate on the key findings and main conclusions.

Chapter 3 provides the answer to the Research Question 1, particularly to the subquestion **(a) What is the best strategy for MAS-based FDIR implementation in small satellites?** The question was addressed by first identifying the required functionalities needed for FDIR: FDI and FIR features. A literature review was conducted to identify methods and techniques that can be applied for control systems.

The proposed agent-based FDIR architecture considered both the functionalities and the methods surveyed to come up with a design that focused on implementation concerns such as maintainability and communication overhead. A trade-off process was implemented to select the strategy that best fitted the needs of small satellites missions. As a result of the trade-off analysis the selected architecture established a fully distributed FDI capabilities while keeping a centralized FIR approach. For the implementation of FDI, model-based methods are more robust than signal-based or data-driven techniques. On the other hand, for fault recovery, data-driven methods, in particular, the statistical techniques are a promising solution for autonomous systems, mostly due to their flexibility of implementation.

Two operation scenarios were implemented using small satellite missions. The case study focused on the components of the ADCS, particularly on the gyroscope performance. The operation scenarios implemented numerical simulations to show the feasibility of the proposed architecture and FDI and FIR methods. Overall, the proposed agent FDIR architecture showed positive results with respect to response time, communication overhead, fault resilience, and maintainability when compared with the other architecture candidates explored.

Chapter 4 was also focused on addressing Research Question 1, particularly subquestions **(b) What are the most critical services that onboard computers shall provide to implement MAS-based software architectures in ADCS computers?** and **(c) How to balance services workload for an efficient MAS-based OBSW architecture implementation?** Communication was identified as the most critical capability required for the design and implementation of agent-based software architecture for satellite systems. The research found that agent communication languages and protocols play a signifi-

cant role in the performance of MAS-based systems. For that purpose, Agent Interaction Protocols (AIP) were studied in detail to understand the advantages of using them as a mean for increasing code modularity and re-use. As the main result of that analysis, a software communication architecture was proposed and described. That architecture can be adopted for its use in satellite systems with distributed data processing on-board.

The second part of that Chapter was devoted to present, implement and characterize a bus load and utilization model for a distributed data bus onboard a satellite. The presented model was analytically conceptualized and compared to discrete time simulation scenarios using the ADCS subsystem as the case study. The simulation results showed an agreement between the analytic and the simulated model for nominal mission conditions. Also, it presented a sensitivity analysis to understand the impact on bus performance of scalability and implementation parameters, such as the sensor sampling period and the bus synchronization period. The chapter also proposed an algorithm that allows balancing the performance of the communication bus by controlling bus configuration parameters.

Chapter 5 addressed Research Question 2 on **How to model the ADCS software architectures for small satellites using a multi-agent systems approach?** For that purpose, a model-based methodology for agent-oriented software was introduced and described in detail. The proposed methodology established a meta-modeling framework and a workflow that specified the activities and steps required to complete an end-to-end software design process for satellite systems.

The proposed meta-model extends state-of-art meta-models to include Fault Detection Isolation and Recovery features by design on MAS-based software. Also, the proposed methodology described a workflow that integrates the analysis, design, implementation and verification activities into a single workflow that enables end-to-end agent-based onboard software development. The methodology was demonstrated by implementing the case study with the ADCS of a satellite. The results showed the feasibility of the methodology for modeling satellite's onboard software. However, the main result of the case study was its utilization for improving ADCS performance by integrating FDIR capabilities with low implementation overhead. The validation of the proposed methodology focused on four critical elements: completeness, consistency, feasibility and testing capabilities.

Finally, Chapter 6 answered Research Question 3 on **How to optimize multi-agent system organization according to ADCS mission requirements and constraints?** In that chapter the main effort was put on obtaining a proper formulation of the optimization problem. That required efforts on translating the requirements and constraints of space software design to interaction constraints in the MAS-based software architecture. Also, it required defining a cost function for the total cost of communication, so that the organization of the software agents minimized the cost communication between agents while satisfying pre-established structures required by design. This Chapter also reviewed the strategies used in MAS design for their organization. It focused on implementing hierarchical and team organizations to provide fixed structures on the system and using randomized search heuristics for achieving a sub-optimal design that minimized the total cost of communication. The optimization problem was solved using customized genetic algorithms that enabled locking fixed interaction genes so that in-

teraction constraints representing mission requirements were satisfied. The simulation case study was focused on demonstrating the advantage of using randomized search for different problem sizes and showing the impact of the network size in the time to find a solution and the number of iterations required for achieving the fitness target.

7.2. INNOVATIONS AND CONTRIBUTIONS

As result of this dissertation the following innovations were accomplished:

1. **An agent-based FDIR software architecture**

   A novel FDIR architecture combining model-based and data-driven techniques was proposed using an agent-based approach. The main advantage of the proposed architecture is its ability to instantly detect the fault of components, and triggering specific recovery features. The proposed architecture also balances out response time, communication overhead, fault resilience, and maintainability, so that the implementation of that architecture is feasible for highly resources constrained systems.

2. **A bus utilization model for distributed data communication buses**

   An analytic model was proposed to describe the bus utilization behavior under nominal conditions for a distributed spacecraft data bus implementing CAN communication protocol. The proposed model took as inputs design parameters such as sensor data collection period, bus synchronization period and data rate. Also, it considered the scalability of the network as part of a sensibility analysis for understanding the performance issues that can be produced as the number of components in the bus increase. The model was verified against a discrete time numerical simulation. The results showed agreement between the analytical model and the discrete simulation model with a difference of less than 5% in the nominal operation region.

3. **A Model-driven methodology for designing fault-tolerant MAS-based software**

   Improving the modeling techniques and artifacts for software eases its implementation and reduces its development time. The meta-model proposed in Chapter 5 extends state-of-the-art MAS-based methodologies to integrate FDIR elements to increase resilience in satellite's onboard software. The proposed methodology offers an end-to-end workflow for the analysis, design and verification and validation phases.

4. **An open software library for agent-based software development in miniaturized satellites**

   The implementation artifacts for the Methodology proposed in Chapter 5 were worked out into an open software library for highly miniaturized systems. As a result, a Multi-Agent Systems for Embedded systems toolkit was released in GitHub as MAES. The library was targeted and tested on the Delfi-PQ micro-controller so that it can be used for the implementation of the onboard software in Pocket-Qube missions.

5. **A genetic algorithm for optimal organization of agent-based software**

   In Chapter 6 the proper formulation of the optimization of total communication cost problem was demonstrated using agent-based architectures with fixed interaction constraints. It allowed a search heuristic that balances exploitation and exploration to find the best solution in the shortest time possible. That algorithm can be also adapted to analyze communication cost in physically distributed systems such as satellite communication constellations.

7.3. RESEARCH OUTLOOK

This thesis focused on addressing the problem of software complexity through the use of agent-based software architectures. That was approached by extending the state-of-art component-based design implemented with object-oriented paradigms to include multi-agent systems concepts. The development of this dissertation required proposing new architectures, models and methodologies to enable the adoption of MAS-based software architectures as a robust approach for designing fault-tolerant OBSW.

Future space missions with small satellites will require advanced software capabilities to boost their performance and enable autonomous fault detection, isolation, and recovery. That requires enabling new computational models and artifacts that can be adopted in the design and development of onboard-software. Besides the new applications that agent-based software architectures support, it is necessary to provide an outlook of their implementation challenges.

7.3.1. NEW APPLICATIONS

This dissertation provides more insight into how agent-based software architectures improve the reliability of space missions. This supports:

1. **Precise Pointing Capabilities:** Agent-based software architectures provide embedded capabilities for proactive FDIR that can be used to improve performance and reliability of missions requiring precise pointing capabilities. Examples were used along the dissertation with laser communication case studies as well as formation flying missions.

2. **Advanced Distributed and Fractionated Space Systems:** The adoption of a distributed computing model based on Multi-Agent Systems supports the development of a novel application with distributed space systems, for instance, satellite constellations, robots swarming and fractionated space systems. That will open new opportunities for space applications that improve its societal relevance, for instance, quality of life of Earth's population (e.g., increasing knowledge about climate change).

3. **Onboard Deep Learning Capabilities:** Machine learning, particularly deep learning is a new tool adopted for increasing performance of autonomous systems such as self-driving cars, or earlier disease detection in medicine. Agent-based architectures enable the possibility to bring deep learning to the onboard software of satellites to open new possibilities for autonomous operations, but mainly for proactive fault-detection and recovery.

4. **Network configuration for Satellite Constellations:** The methodology, models and algorithms presented as results of this dissertation can be adapted and applied on the implementation of the network configuration of satellite constellations for optimizing communication cost and improving fault resilience.

7.3.2. IMPLEMENTATION ASPECTS

The adoption of a new computational model for satellite's onboard software comes with several challenges related mainly to the qualification of agent-based software architectures for space systems operations. That will require an incremental approach to qualify the software libraries and methodologies using technology demonstration missions.

7.4. RECOMMENDATIONS

This dissertation addressed research activities related to the adoption of agent-based software architectures to design fault-tolerant software for small satellites. However, there is a list of activities to complement the results presented in this thesis:

1. **Demonstrate the proposed agent-based FDIR architecture in a satellite mission**.

   Using the library co-developed with Chan-Zheng (2017) as a spin-off of this dissertation, it is feasible to implement the proposed agent-based FDIR architecture in a mission with nano-satellites or pico-satellites. Currently, the library is supported for the hardware and software stack of Delfi-PQ. It is recommended to port the library and add features for nano-satellite missions. That development will be carried out by the Space Systems Laboratory at the Costa Rica Institute of Technology (TEC).

2. **Develop a testbed for validating the proposed bus utilization model for CAN**

   During this research, an analytic and a simulation model was developed to characterize the busload of a CAN bus for distributed command and control data buses of satellites. This model required more validation so that an initial set of experiments was conducted by Orsel (2016). A further development of this testbed is of interest since it enables a more tailored design of the communication capabilities onboard satellites, but also will allow more in-depth experiments to be carried on data bus performance.

3. **Adapt the MASSA methodology to use it with Capella Modeling tools**

   Originally, the Multi-Agents Systems for Satellite Applications methodology was planned for its implementation using SysML artifacts and following the workflow proposed in Chapter 5. Recent developments at ESA show the potential of using the Arcadia model and its implementation tools such as $Capella^{TM}$ for Model-based systems engineering of spacecraft. There is a recommendation on developing activities to match and migrate MASSA to the Arcadia meta-model, so there are more support and adoption opportunities from the space community for the features and innovations proposed by MASSA. Also, model-to-model conversion artifacts can be implemented as the ones developed in the work of der Gaag (2017).

4. **Implement comparative search heuristics such as PSO and Ant Colony Optimization for the organization of agent-based software.**

   As a result of the trade-off analysis and due to time constraints, the implementation of the total cost optimization algorithm focused on the use of Genetic Algorithms. The results were compared with the Brute Force Algorithm. There is room for improvement so that other search heuristics can be implemented for comparison purposes. For instance, a PSO can be carried out or implementing Ant colony optimization algorithms. This development can be for instance followed up as a Master thesis in Computer Sciences.

REFERENCES

Alghamdi, M. I., Jiang, X., Zhang, J., Zhang, J., Jiang, M., and Qin, X. (2017). Towards two-phase scheduling of real-time applications in distributed systems. *Journal of Network and Computer Applications*, 84:109 – 117.

Aljarah, I., Faris, H., and Mirjalili, S. (2018). Optimizing connection weights in neural networks using the whale optimization algorithm. *Soft Computing*, 22(1):1–15.

Antonova, I. and Batchkova, I. (2008). Development of multi-agent control systems using uml/sysml. In *2008 4th International IEEE Conference Intelligent Systems*, volume 1, pages 6–26. IEEE.

Anwar, S. and Niu, W. (2014). A nonlinear observer based analytical redundancy for predictive fault tolerant control of a steer-by-wire system. *Asian Journal of Control*, 16(2):321–334.

Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V., and Rebollo, M. (2011). An abstract architecture for virtual organizations: The thomas approach. *Knowledge and Information Systems*, 29(2):379–403.

Arruego, I., Guerrero, M., Rodriguez, S., Martinez-Oter, J., Jiménez, J. J., Dominguez, J. A., Martín-Ortega, A., De Mingo, J., Rivas, J., Apestigue, V., et al. (2009). Owls: A ten-year history in optical wireless links for intra-satellite communications. *IEEE Journal on selected areas in communications*, 27(9).

Artursson, T., Eklöv, T., Lundström, I., Mårtensson, P., Sjöström, M., and Holmberg, M. (2000). Drift correction for gas sensors using multivariate methods. *Journal of chemometrics*, 14(5-6):711–723.

Atkinson, C. and Kuhne, T. (2003). Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41.

Atkinson, C. and Kühne, T. (2008). Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359.

Auger, A. and Doerr, B. (2011). *Theory of randomized search heuristics: Foundations and recent developments*, volume 1. World Scientific.

Baheti, R. and Gill, H. (2011). Cyber-physical systems. *The impact of control technology*, 12:161–166.

Bak, T. (1999). *Spacecraft attitude determination: A magnetometer approach*. PhD thesis, Aalborg Universitetsforlag.

Banker, R. D., Datar, S. M., Kemerer, C. F., and Zweig, D. (1993). Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–95.

Barnhart, D. J., Vladimirova, T., Baker, A. M., and Sweeting, M. N. (2009a). A low-cost femtosatellite to enable distributed space missions. *Acta Astronautica*, 64(11-12):1123–1143.

Barnhart, D. J., Vladimirova, T., and Sweeting, M. N. (2007). Very-small-satellite design for distributed space missions. *Journal of Spacecraft and Rockets*, 44(6):1294–1306.

Barnhart, D. J., Vladimirova, T., and Sweeting, M. N. (2009b). Satellite miniaturization techniques for space sensor networks. *Journal of Spacecraft and Rockets*, 46(2):469.

Basili, V. R. and Perricone, B. T. (1984). Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52.

Bekkeng, J. (2009). Calibration of a novel mems inertial reference unit. *IEEE Transactions on Instrumentation and Measurement*, 58(6):1967–1974.

Bellantoni, J. and Dodge, K. (1967). A square root formulation of the kalman-schmidt filter. *AIAA journal*, 5(7):1309–1314.

Bellifemine, F., Caire, G., Vitaglione, G., Rimassa, G., and Greenwood, D. (2005). *The JADE Platform and Experiences with Mobile MAS Applications*, pages 1–20. Birkhäuser Basel, Basel.

Bellifemine, F., Poggi, A., and Rimassa, G. (2000). Developing multi-agent systems with jade. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 89–103. Springer.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons.

Belward, A. S. and Skøien, J. O. (2015). Who launched what, when and why; trends in global land cover observation capacity from civilian earth observation satellites. *ISPRS Journal of Photogrammetry and Remote Sensing*, 103:115–128.

Béounes, C., Aguéra, M., Arlat, J., Bachmann, S., Bourdeau, C., Doucet, J.-E., Kanoun, K., Laprie, J.-C., Metge, S., de Souza, J. M., et al. (1993). Surf-2: A program for dependability evaluation of complex hardware and software systems. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 668–673. IEEE.

Bergenti, F., Caire, G., and Gotta, D. (2014). Agents on the move: Jade for android devices. In *WOA*, volume 1260.

Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., and Yushtein, Y. (2014a). An integrated process for fdir design in aerospace. In *International Symposium on Model-Based Safety and Assessmemt*, pages 82–95. Springer.

Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., and Yushtein, Y. (2014b). An integrated process for FDIR design in aerospace. In Ortmeier, F. and Rauzy, A., editors, *Model-Based Safety and Assessment*, volume 8822 of *Lecture Notes in Computer Science*, pages 82–95. Springer International Publishing.

Blickle, T. and Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394.

Bo, Z., Bo, D., and Yuanchun, L. (2011). Support vector machine observer based fault detection for reconfigurable manipulators. *30th Chinese Control Conference (CCC)*, pages 3979–3984.

Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75.

Borde, J., Teston, F., Santandrea, S., and Boulade, S. (2004). Feasibility of the proba 3 formation flying demonstration mission as a pair of microsats in gto. In *Small Satellites, Systems and Services*, volume 571.

Bouwmeester, J., Brouwer, G., Gill, E., Monna, G., and Rotteveel, J. (2010). Design status of the delfi-next nanosatellite project. In *61st International Astronautical Congress, Prague, Czech Republic, 27 September-1 October 2010*. International Astronautical Federation.

Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., and Roveri, M. (2009). The compass approach: Correctness, modelling and performability of aerospace systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 173–186. Springer.

Bravo, M., Silva-L, B., et al. (2015). Multi-agent communication heterogeneity. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 583–588. IEEE.

Broster, I. and Burns, A. (2001). Timely use of the CAN protocol in critical hard real-time systems with faults. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 95–102. IEEE.

Brouwer, D. and Clemence, G. M. (2013). *Methods of celestial mechanics*. Elsevier.

Burns, A. and McDermid, J. A. (1994). Real-time safety-critical systems: analysis and synthesis. *Software Engineering Journal*, 9(6):267–281.

Butcher, J. (2007). Runge-kutta methods. *Scholarpedia*, 2(9):3147.

Cacciatore, F., Sánchez, R., Agenjo, A., Puente, N., Ardura, C., Olier, L., Gómez, V., Saponara, M., and Saavedra, G. (2016). Rapid deployment of design environment for euclid AOCS design. *6th International Conference on Astrodynamics Tools and Techniques*.

Calvaresi, D., Sernani, P., Marinoni, M., Claudi, A., Balsini, A., Dragoni, A. F., and Buttazzo, G. (2016). A framework based on real-time os and multi-agents for intelligent autonomous robot competitions. In *Industrial Embedded Systems (SIES), 2016 11th IEEE Symposium on*, pages 1–10. IEEE.

Cederman, D., Hellstrom, D., Sherrill, J., Bloom, G., Patte, M., and Zulianello, M. (2014). Rtems smp for leon3/leon4 multi-processor devicies. In *DASIA 2014-DAta Systems In Aerospace*, volume 725.

Chan-Zheng, C. (2017). MAES: A multi-agent systems framework for embedded systems. Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology, the Netherlands.

Chaves-Jiménez, A., Guo, J., and Gill, E. (2017). Impact of atmospheric coupling between orbit and attitude in relative dynamics observability. *Journal of Guidance, Control, and Dynamics*, pages 1–8.

Chen, J. and Patton, R. (1999). Basic principles of model-based fault diagnosis. In *Robust Model-Based Fault Diagnosis for Dynamic Systems*, volume 3 of *The International Series on Asian Studies in Computer and Information Science*, pages 19–64. Springer US.

Chen, J. J.-Y. and Su, S.-W. (2003). Agentgateway: A communication tool for multi-agent systems. *Information Sciences*, 150(3):153 – 164. Internet Computing.

Chien, S., Sherwood, R., Burl, M., Knight, R., Rabideau, G., Engelhardt, B., Davies, A., Zetocha, P., Wainright, R., Klupar, P., et al. (2014). A demonstration of robust planning and scheduling in the techsat-21 autonomous sciencecraft constellation. In *Sixth European Conference on Planning*.

Chopra, A. K., Artikis, A., Bentahar, J., Colombetti, M., Dignum, F., Fornara, N., Jones, A. J. I., Singh, M. P., and Yolum, P. (2013). Research directions in agent communication. *ACM Trans. Intell. Syst. Technol.*, 4(2):20:1–20:23.

Chu, J., Guo, J., and Gill, E. K. (2013). Fractionated space infrastructure for long-term earth observation missions. In *Aerospace Conference, 2013 IEEE*, pages 1–9. IEEE.

Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing*, 36(3):287 – 314. Higher Order Statistics.

Cooper, A. E. and Chow, W. T. (1976). Development of on-board space computer systems. *IBM Journal of Research and Development*, 20(1):5–19.

Crassidis, J. L., Markley, F. L., and Cheng, Y. (2007). Survey of nonlinear attitude estimation methods. *Journal of guidance control and dynamics*, 30(1):12.

Curey, R. K., Ash, M. E., Thielman, L. O., and Barker, C. H. (2004). Proposed ieee inertial systems terminology standard and other inertial sensor standards. In *Position Location and Navigation Symposium, 2004. PLANS 2004*, pages 83–90. IEEE.

da Silva, V. T. and de Lucena, C. J. (2007). Modeling multi-agent systems. *Communications of the ACM*, 50(5):103–108.

D'Angelo, G., Tipaldi, M., Glielmo, L., and Rampone, S. (2017). Spacecraft autonomy modeled via markov decision process and associative rule-based machine learning. In *Metrology for AeroSpace (MetroAeroSpace), 2017 IEEE International Workshop on*, pages 324–329. IEEE.

Davis, L. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold.

Degueule, T., Combemale, B., and Jézéquel, J.-M. (2017). On Language Interfaces. In Meyer, B. and Mazzara, M., editors, *PAUSE: Present And Ulterior Software Engineering*, pages 1–100. Springer.

Deloach, S. (2004). The mase methodology. *Methodologies and software engineering for agent systems*, pages 107–125.

der Gaag, J. V. (2017). SysML to SLIM transformation methodology. Master's thesis, Faculty of Aerospace Engineering at Delft University of Technology, the Netherlands.

do Nascimento, N. M. and de Lucena, C. J. P. (2017). Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. *Information Sciences*, 378:161–176.

Doncaster, B., Shulman, J., Bradford, J., and Olds, J. (2016). Spaceworks' 2016 nano/microsatellite market forcast. In *AIAA/USU Conference on Small Satellites*.

Dou, C., Yue, D., Han, Q.-L., and Guerrero, J. M. (2017). Multi-agent system-based event-triggered hybrid control scheme for energy internet. *IEEE Access*, 5:3263–3272.

Dvorak, D. L. and Lyu, M. (2009). Nasa study on flight software complexity. *NASA office of chief engineer*.

Eickhoff, J. (2011). *Onboard Computers, Onboard Software and Satellite Operations: An Introduction*. Springer Science & Business Media.

El Faouzi, N.-E., Leung, H., and Kurian, A. (2011). Data fusion in intelligent transportation systems: Progress and challenges–a survey. *Information Fusion*, 12(1):4–10.

El-Sheimy, N., Hou, H., and Niu, X. (2008). Analysis and modeling of inertial sensors using allan variance. *IEEE Transactions on Instrumentation and Measurement*, 57(1):140–149.

Elsenbroich, C. and Gilbert, N. (2014). Agent-based modelling. In *Modelling norms*, pages 65–84. Springer.

Etkin, B. and Hughes, P. (1967). Explanation of the anomalous spin behavior of satellites with long, flexible antennae. *Journal of Spacecraft and Rockets*, 4(9):1139–1145.

Fayyaz, M., Vladimirova, T., and Caujolle, J.-M. (2012). Adaptive middleware design for satellite fault-tolerant distributed computing. In *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, pages 23–30. IEEE.

Feiler, P. H. (2010). Model-based validation of safety-critical embedded systems. In *Aerospace Conference, 2010 IEEE*, pages 1–10. IEEE.

Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 128–135. IEEE.

Ferber, J., Gutknecht, O., and Michel, F. (2003). From agents to organizations: an organizational view of multi-agent systems. In *International Workshop on Agent-Oriented Software Engineering*, pages 214–230. Springer.

Field, M. and Pence, D. (1984). Spacecraft attitude, rotations and quaternions. *UMAP Modules*, 5(2):130.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, pages 456–463, New York, NY, USA. ACM.

FIPA (2002a). Agent message transport service specification. Webpage.

FIPA (2002b). Fipa agent message transport service specification. `http://www.fipa.org/specs/fipa00067/`.

Fipa, A. (2002). FIPA ACL message structure specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

Fonseca, S., Griss, M., and Letsinger, R. (2001). Evaluation of the zeus mas framework. In *Second International Workshop in Software Agents and Workflows for Systems Interoperability*.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Franklin, G. F., Powell, J. D., Emami-Naeini, A., and Powell, J. D. (1994). *Feedback control of dynamic systems*, volume 3. Addison-Wesley Reading, MA.

Furano, G. and Menicucci, A. (2018). Roadmap for on-board processing and data handling systems in space. In *Dependable Multicore Architectures at Nanoscale*, pages 253–281. Springer.

Gaisler, J. (2002). A portable and fault-tolerant microprocessor based on the sparc v8 architecture. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 409–415. IEEE.

Gangl, E. C. (2013). A case study on us government military standard development. *IEEE Aerospace and Electronic Systems Magazine*, 28(7):40–45.

Gao, S., Clark, K., Unwin, M., Zackrisson, J., Shiroma, W., Akagi, J., Maynard, K., Garner, P., Boccia, L., Amendola, G., et al. (2009). Antennas for modern small satellites. *IEEE Antennas and Propagation Magazine*, 51(4).

Gao, Z., Cecati, C., and Ding, S. (2015a). A survey of fault diagnosis and fault-tolerant techniques; part II: Fault diagnosis with knowledge-based and hybrid/active approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3768–3774.

Gao, Z., Cecati, C., and Ding, S. X. (2015b). A survey of fault diagnosis and fault-tolerant techniques;part i: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3757–3767.

Garcia, A., Sant'Anna, C., Chavez, C., da Silva, V., de Lucena, C., and von Staa, A. (2004). Separation of concerns in multi-agent systems: An empirical study. *Software Engineering for Multi-Agent Systems II*, pages 343–344.

Gascueña, J. M., Navarro, E., and Fernández-Caballero, A. (2012). Model-driven engineering techniques for the development of multi-agent systems. *Engineering Applications of Artificial Intelligence*, 25(1):159–173.

Gasser, L. (2000). Mas infrastructure: Definitions, needs and prospects. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, volume 1887, pages 1–11. Springer.

Ge, W., Wang, J., Zhou, J., Wu, H., and Jin, Q. (2015). Incipient fault detection based on fault extraction and residual evaluation. *Industrial & Engineering Chemistry Research*, 54(14):3664–3677.

George, A. D. and Wilson, C. M. (2018). Onboard processing with hybrid and reconfigurable computing on small satellites. *Proceedings of the IEEE*, 106(3):458–470.

Gill, E., Montenbruck, O., and Kayal, H. (2001). The bird satellite mission as a milestone toward gps-based autonomous navigation. *Navigation*, 48(2):69–75.

Gill, E., Sundaramoorthy, P., Bouwmeester, J., Zandbergen, B., and Reinhard, R. (2013). Formation Flying within a constellation of nano-satellites: The QB50 mission. *Acta Astronautica*, 82(1):110–117.

Glavic, M. (2006). Agents and multi-agent systems: a short introduction for power engineers. Technical Report 1, University of Liege, Sart-Tilman B-28, 4000 Liege, BELGIUM.

González-Potes, A., Mata-López, W. A., Ibarra-Junquera, V., Ochoa-Brust, A. M., Martínez-Castro, D., and Crespo, A. (2016). Distributed multi-agent architecture for real-time wireless control networks of multiple plants. *Engineering Applications of Artificial Intelligence*, 56:142–156.

Gorbenko, A. and Popov, V. (2012). Task-resource scheduling problem. *International Journal of Automation and Computing*, 9(4):429–441.

Gregori, M. E., Cámara, J. P., and Bada, G. A. (2006a). A jabber-based multi-agent system platform. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1282–1284, New York, NY, USA. ACM.

Gregori, M. E., Cámara, J. P., and Bada, G. A. (2006b). A jabber-based multi-agent system platform. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1282–1284. ACM.

Grillmayer, G., Hirth, M., Huber, F., and Wolter, V. (2006). Development of an FPGA based attitude control system for a micro-satellite. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Keystone, Colorado*.

Großekatthöfer, K. and Yoon, Z. (2012). Introduction into quaternions for spacecraft attitude representation. *TU Berlin*, 16.

Guerrieri, D. C., Cervone, A., and Gill, E. (2016). Analysis of nonisothermal rarefied gas flow in diverging microchannels for low-pressure microresistojets. *Journal of Heat Transfer*, 138(11):112403.

Guo, J., Bouwmeester, J., and Gill, E. (2016). In-orbit results of Delfi-n3Xt: Lessons learned and move forward. *Acta Astronautica*, 121:39–50.

Guo, J. and Gill, E. (2013). Delffi: Formation flying of two cubesats for technology, education and science. *International Journal of Space Science and Engineering*, 1(2):113–127.

Gupta, A. (2006). Hotelling's t-squared statistic. In *Encyclopedia of Environmetrics*. John Wiley & Sons, Ltd.

Guruprasad, S., Bisnath, S., Lee, R., and Kozinski, J. (2016). Design and implementation of a low-cost soc-based software gnss receiver. *IEEE Aerospace and Electronic Systems Magazine*, 31(4):14–19.

Hahn, C., Madrigal-Mora, C., and Fischer, K. (2009). A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266.

Hassan, R., Cohanim, B., de Weck, O., and Venter, G. (2004). A copmarison of particle swarm optimization and the genetic algorithm. *American Institute of Aeronautics and Astronautics*.

Hassani, K. and Lee, W.-S. (2015). An intelligent architecture for autonomous virtual agents inspired by onboard autonomy. In *Intelligent Systems' 2014*, pages 391–402. Springer.

Hayden, S., Carrick, C., Yang, Q., et al. (1999). Architectural design patterns for multiagent coordination. In *Proceedings of the International Conference on Agent Systems*, volume 99.

He, W., Zi, Y., Chen, B., Wu, F., and He, Z. (2015). Automatic fault feature extraction of mechanical anomaly on induction motor bearing using ensemble super-wavelet transform. *Mechanical Systems and Signal Processing*, 54–55:457 – 480.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243.

Hedin, A. E. (1991). Extension of the msis thermosphere model into the middle and lower atmosphere. *Journal of Geophysical Research: Space Physics*, 96(A2):1159–1172.

Heidt, M. H., Puig-Suari, P. J., Moore, A. S., Nakasuka, S., Twiggs, R. J., and Moore, A. S. (2000). Cubesat : A new generation of picosatellite for education and industry low-cost space experimentation. In *15th Annual/USU Conference on Small Satellites*.

Hili, N., Dingel, J., and Beaulieu, A. (2017). Modelling and code generation for real-time embedded systems with uml-rt and papyrus-rt. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 509–510. IEEE Press.

Hinchey, M. and Vassev, E. (2012). *Multi-agent systems–Theory, approaches and NASA applications*, volume 32, chapter 7, pages 181–202. IOS Press Ebooks.

Hitt, D., Robinson, K. F., and Creech, S. D. (2016). NASA's space launch system: A new opportunity for cubesats. In *5th Interplanetary CubeSat Workshop*.

Hubbard, S. (2014). Cubesats, return on investment, deep space, and physics.

Hughes, P. C. (2012). *Spacecraft attitude dynamics*. Courier Corporation.

Hwang, I., Kim, S., Kim, Y., and Seah, C. (2010). A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18(3):636–653.

Hyvarinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, 10(3):626–634.

Isermann, R. (2006). Fault detection with state observers and state estimation. *in Fault-Diagnosis Systems, Springer Berlin Heidelberg*, pages 231–252.

Janson, S. and Welle, R. (2014a). The NASA optical communication and sensor demonstration program: An update. In *28th Annual AIAA/USU Conference on Small Satellites*.

Janson, S. W. and Welle, R. P. (2014b). The nasa optical communication and sensor demonstration program: an update. In *28th Annual AIAA/USU Conference on Small Satellites*, pages 4–7.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial intelligence*, 117(2):277–296.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41.

Jensen, K. F. and Vinther, K. (2010). Attitude determination and control system for AAUSAT3. Master's thesis, Department of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7, Aalborg, Denmark.

Jiang, T., Khorasani, K., and Tafazoli, S. (2008). Parameter estimation-based fault detection, isolation and recovery for nonlinear satellite models. *IEEE Transactions on control systems technology*, 16(4):799–808.

Jin, Z. and Murray, R. M. (2006). Multi-hop relay protocols for fast consensus seeking. In *Decision and Control, 2006 45th IEEE Conference on*, pages 1001–1006. IEEE.

Juan, T., Pearce, A., and Sterling, L. (2002). Roadmap: extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 3–10. ACM.

Kachitvichyanukul, V. (2012). Comparison of three evolutionary algorithms: Ga, pso, and de. *Industrial Engineering and Management Systems*, 11(3):215–223.

Kang, S.-M. and Ahn, H.-S. (2016). Design and realization of distributed adaptive formation control law for multi-agent systems with moving leader. *IEEE Transactions on Industrial Electronics*, 63(2):1268–1279.

Kaslow, D., Anderson, L., Asundi, S., Ayres, B., Iwata, C., Shiotani, B., and Thompson, R. (2015). Developing a cubesat model-based system engineering (mbse) reference model-interim status. In *2015 IEEE Aerospace Conference*, pages 1–16. IEEE.

Katzela, I., Bouloutas, A. T., and Calo, S. B. (1995). Centralized vs distributed fault localization. In *International Symposium on Integrated Network Management*, pages 250–261. Springer.

Keyes, R. W. (2006). The impact of moore's law. *IEEE solid-state circuits society newsletter*, 20(3):25–27.

Khalastchi, E. and Kalech, M. (2018). On fault detection and diagnosis in robotic systems. *ACM Computing Surveys (CSUR)*, 51(1):9.

Khaleghi, B., Khamis, A., Karray, F. O., and Razavi, S. N. (2013). Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44.

Khurram, M. and Zaidi, S. M. Y. (2005). CAN as a spacecraft communication bus in LEO satellite mission. In *Proceedings of 2nd International Conference on Recent Advances in Space Technologies, 2005. RAST 2005.*, pages 432–437.

Kim, K. (2011). Analysis of hysteresis for attitude control of a microsatellite. *San Jose State University, http://www. engr. sjsu. edu/spartnik/adac. html*.

Kimm, H. and Jarrell, M. (2014). Controller area network for fault tolerant small satellite system design. In *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, pages 81–86. IEEE.

Klinar, W. J., Saldana, R. L., Kubiak, E. T., Smith Jr, E. E., Peters, W. H., and Stegall, H. W. (1975). Space shuttle flight control system. *IFAC Proceedings Volumes*, 8(1):302–310.

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

Kuwahara, T., Böhringer, F., Falke, A., Eickhoff, J., Huber, F., and Roser, H.-P. (2009). Fpga-based operational concept and payload data processing for the flying laptop satellite. *Acta Astronautica*, 65(11–12):1616 – 1627.

Kwok, Y.-K. and Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471.

Lalanda, P., McCann, J. A., and Diaconescu, A. (2013). *Autonomic Computing*. Springer.

Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, 32(4):18–25.

Laouadi, M. A., Mokhati, F., and Seridi, H. (2014). A novel organizational model for real time mas: Towards a formal specification. In *Intelligent Systems for Science and Information*, pages 171–180. Springer International Publishing.

Lawler, E. L. (1976). *Combinatorial optimization: networks and matroids*. Courier Corporation.

Lawrenz, W. (1997). CAN system engineering. *From theory to practical applications, New York*.

Le Vinh, T., Bouzefrane, S., Farinone, J.-M., Attar, A., and Kennedy, B. P. (2015). Middleware to integrate mobile devices, sensors and cloud computing. *Procedia Computer Science*, 52:234–243.

Lee, E. A. (2008). Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, volume 00, pages 363–369.

Lefferts, E. J., Markley, F. L., and Shuster, M. D. (1982). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429.

Li, J., Xiong, K., Wei, X., and Zhang, G. (2017a). A star tracker on-orbit calibration method based on vector pattern match. *Review of Scientific Instruments*, 88(4):043101.

Li, S. and Kokar, M. M. (2013). *Agent Communication Language*, pages 7–44. Springer New York, New York, NY.

Li, T., Shen, H., Yuan, Q., Zhang, X., and Zhang, L. (2017b). Estimating ground-level pm2. 5 by fusing satellite and station observations: A geo-intelligent deep learning approach. *Geophysical Research Letters*, 44(23).

Li, Z., Liu, G., Zhang, R., and Zhu, Z. (2011). Fault detection, identification and reconstruction for gyroscope in satellite based on independent component analysis. *Acta Astronautica*, 68(7–8):1015–1023.

Liang, H., Su, H., Wang, X., and Chen, M. Z. (2016). Swarming of heterogeneous multi-agent systems with periodically intermittent control. *Neurocomputing*, 207:213–219.

Liu, B. and Iwamura, K. (2000). Topological optimization models for communication network with multiple reliability goals. *Computers & Mathematics with Applications*, 39(7-8):59–69.

Llorente, J., Agenjo, A., Carrascosa, C., De Negueruela, C., Mestreau-Garreau, A., Cropp, A., and Santovincenzo, A. (2013). Proba-3: Precise formation flying demonstration mission. *Acta Astronautica*, 82(1):38–46.

Long, B., Jiang, X.-W., and Song, Z.-J. (2005). Real-time monitoring and diagnosis technology for satellite telemetry data based on multi-agent. *Acta Aeronautica Et Astronautica Sinica*, 26(6):726–732.

Lützenberger, M., Küster, T., Konnerth, T., Thiele, A., Masuch, N., Heßler, A., Keiser, J., Burkhardt, M., Kaiser, S., and Albayrak, S. (2013). Jiac v: A mas framework for industrial applications. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1189–1190. International Foundation for Autonomous Agents and Multiagent Systems.

Ma, Y., Guo, S., Lin, S., and Cai, Y. (2017). A consensus-based filtering algorithm with state constraints. In *Control Conference (CCC), 2017 36th Chinese*, pages 5502–5506. IEEE.

Macdonald, M. and Badescu, V. (2014). *The international handbook of space technology*. Springer.

Macmillan, S. and Maus, S. (2005). International geomagnetic reference field-the tenth generation. *Earth, planets and space*, 57(12):1135–1140.

Magee, J. and Kramer, J. (1996). Dynamic structure in software architectures. In *ACM SIGSOFT Software Engineering Notes*, volume 21, pages 3–14. ACM.

Maheshwarappa, M. R., Bowyer, M., and Bridges, C. P. (2015). Software defined radio (sdr) architecture to support multi-satellite communications. In *Aerospace Conference, 2015 IEEE*, pages 1–10. IEEE.

Marshall, A. W. and Olkin, I. (1985). A family of bivariate distributions generated by the bivariate bernoulli distribution. *Journal of the American Statistical Association*, 80(390):332–338.

Marzat, J., Piet-Lahanier, H., Damongeot, F., and Walter, E. (2012). Model-based fault diagnosis for aerospace systems: a survey. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of aerospace engineering*, 226(10):1329–1360.

May, R. D. and Loparo, K. A. (2014). The use of software agents for autonomous control of a dc space power system. *IEEE EnergyTech*, pages 28–30.

Mens, T., Magee, J., and Rumpe, B. (2010). Evolving software architecture descriptions of critical systems. *Computer*, 43(5):42–48.

Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212.

Moghaddam, S. H. and Jovanovic, M. R. (2017). Topology design for stochastically-forced consensus networks. *IEEE Transactions on Control of Network Systems*.

Montenbruck, O. and Gill, E. (2012). *Satellite orbits: models, methods and applications*. Springer Science & Business Media.

Nandi, S., Ilamparithi, T., Lee, S. B., and Hyun, D. (2011). Detection of eccentricity faults in induction machines based on nameplate parameters. *IEEE Transactions on Industrial Electronics*, 58(5):1673–1683.

Nguyen, P. T., Schau, V., and Rossak, W. (2011). Performance comparison of some message transport protocol implementations for agent community communication. In *IICS*, pages 193–204.

Nguyen, T., Riesing, K., Kingsbury, R., and Cahoy, K. (2015). Development of a pointing, acquisition, and tracking system for a cubesat optical communication module. In *Proceedings of SPIE–the Society of Photo-Optical Instrumentation Engineers*. SPIE.

Norman, T. J., Jennings, N. R., Faratin, P., and Mamdani, E. (1996). Designing and implementing a multi-agent architecture for business process management. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 261–275. Springer.

Normand, V. and Exertier, D. (2004). Model-driven systems engineering: Sysml & the mdsyse approach at thales. *Ecole d'été CEA-ENSIETA, Brest, France*.

Olfati-Saber, R., Fax, J. A., and Murray, R. M. (2007a). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233.

Olfati-Saber, R., Fax, J. A., and Murray, R. M. (2007b). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233.

Orsel, E. G. A. (2016). Power modelling and optimisation of a communication bus for small satellite missions. Master's thesis, Faculty of Aerospace Engineering, Delft University of Technology.

Oviedo, D., Romero-Ternero, M. d. C., Hernández, M., Carrasco, A., Sivianes, F., and Escudero, J. (2010). Architecture for multiagent-based control systems. In *Distributed Computing and Artificial Intelligence*, pages 97–104. Springer.

Padgham, L., Thangarajah, J., and Winikoff, M. (2014). Prometheus research directions. In *Agent-Oriented Software Engineering*, pages 155–171. Springer.

Padgham, L. and Winikoff, M. (2002). Prometheus: A methodology for developing intelligent agents. In *International Workshop on Agent-Oriented Software Engineering*, pages 174–185. Springer.

Paige, R. F., Matragkas, N., and Rose, L. M. (2016). Evolving models in model-driven engineering: State-of-the-art and future challenges. *Journal of Systems and Software*, 111:272–280.

Panda, S. and Padhy, N. P. (2008). Comparison of particle swarm optimization and genetic algorithm for facts-based controller design. *Applied Soft Computing*, 8(4):1418 – 1427. Soft Computing for Dynamic Data Mining.

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE.

Patel, R. and Rajawat, A. (2013). Recent trends in embedded system software performance estimation. *Design Automation for Embedded Systems*, 17(1):193–213.

Pavón, J. and Gómez-Sanz, J. (2003). Agent oriented software engineering with ingenias. In *International Central and Eastern European Conference on Multi-Agent Systems*, pages 394–403. Springer.

Pirmoradi, F., Sassani, F., and de Silva, C. (2009). Fault detection and diagnosis in a spacecraft attitude determination system. *Acta Astronautica*, 65(5–6):710—-729.

Plummer, C., Roos, P., and Stagnaro, L. (2003). Can bus as a spacecraft onboard bus. In *DASIA 2003-Data Systems In Aerospace*, volume 532.

Puig-Suari, J., Turner, C., and Ahlgren, W. (2001). Development of the standard cubesat deployer and a cubesat class picosatellite. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 1, pages 1–347. IEEE.

Rasmussen, R. and Litty, E. (1981). A voyager attitude control perspective on fault tolerant systems. In *Guidance and Control Conference*, page 1812.

Rauch, H. E., Tung, F., Striebel, C. T., et al. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450.

Ren, W. (2010). Distributed cooperative attitude synchronization and tracking for multiple rigid bodies. *IEEE Transactions on Control Systems Technology*, 18(2):383–392.

Rivas, J. M., Gutiérrez, J. J., Aldea, M., Cuevas, C., Harbour, M. G., Drake, J. M., Medina, J. L., Rioux, L., Henia, R., and Sordon, N. (2016). An experience integrating response-time analysis and optimization with an mde strategy. In *Federation of International Conferences on Software Technologies: Applications and Foundations*, pages 303–316. Springer.

Roberge, V., Tarbouchi, M., and Labonté, G. (2013). Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning. *IEEE Transactions on Industrial Informatics*, 9(1):132–141.

Robertson, R. E. (1958). Gravitational torque on a satellite vehicle. *Journal of the Franklin Institute*, 265(1):13–22.

Rousset, A., Herrmann, B., Lang, C., and Philippe, L. (2014). A survey on parallel and distributed multi-agent systems. In *Padabs 2014, 2nd Workshop on Parallel and Distributed Agent-Based Simulations, in conjunction with Euro-Par 2014*, volume 8805, pages 371–382. Springer.

Runeson, P. and Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131.

Russell, S., Norvig, P., and Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:31.

Saleh, J. H., Hastings, D. E., and Newman, D. J. (2003). Flexibility in system design and implications for aerospace systems. *Acta astronautica*, 53(12):927–944.

Samara, P., Fouskitakis, G., Sakellariou, J., and Fassois, S. (2008). A statistical method for the detection of sensor abrupt faults in aircraft control systems. *IEEE Transactions on Control Systems Technology*, 16(4):789–798.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Sarode, S. and Patil, A. (2016). Implementation of fault tolerant soft processor on fpga. *IJAR*, 2(1):781–784.

Sayed, A. H. et al. (2014). Adaptation, learning, and optimization over networks. *Foundations and Trends® in Machine Learning*, 7(4-5):311–801.

Schaus, V., Fischer, P. M., Lüdtke, D., Braukhane, A., Romberg, O., and Gerndt, A. (2010). Concurrent engineering software development at german aerospace center-status and outlook. In *4th International Workshop on System & Concurrent Engineering for Space Applications*.

Scholz, A., Hsiao, T.-H., Juang, J.-N., and Cherciu, C. (2017). Open source implementation of ecss can bus protocol for cubesats. *Advances in Space Research*.

Serway, R. A. and Jewett, J. W. (2018). *Physics for scientists and engineers with modern physics*. Cengage learning.

Shehory, O. and Sturm, A. (2014). Multi-agent systems: A software architecture viewpoint. In *Agent-Oriented Software Engineering*, pages 57–78. Springer.

Sherwood, R., Chien, S., Burl, M., Knight, R., Rabideau, G., Engelhardt, B., Davies, A., Williams, B., Greeley, R., Baker, V., et al. (2001). The techsat 21 autonomous sciencecraft constellation demonstration. In *International Symposium on Artificial Intelligence Robotics and Automation in Space*.

Shi, Y. and Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 101–106. IEEE.

Shrivastava, S. and Modi, V. (1983). Satellite attitude dynamics and control in the presence of environmental torques- a brief survey. *Journal of Guidance, Control, and Dynamics(ISSN 0731-5090)*, 6:461–471.

Sidi, M. J. (1997). *Spacecraft dynamics and control: a practical engineering approach*, volume 7. Cambridge university press.

Simon, D. (2006). *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons.

Sinopoli, B., Schenato, L., Franceschetti, M., Poolla, K., Jordan, M. I., and Sastry, S. S. (2004). Kalman filtering with intermittent observations. *IEEE transactions on Automatic Control*, 49(9):1453–1464.

Sonnek, J., Greensky, J., Reutiman, R., and Chandra, A. (2010). Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 228–237. IEEE.

Speretta, S., Pérez Soriano, T., Bouwmeester, J., Carvajal Godínez, J., Watts, T., Menicucci, A., Sundaramoorthy, P., Guo, J., and Gill, E. (2016). Cubesats to pocketqubes: Opportunities and challenges. In *Proceedings of the 67th International Astronautical Congress (IAC)*. IAF.

Stockwell, W. (2003). Angle random walk. *Application Note. Crossbow Technologies Inc*, pages 1–4.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

Sturm, A. and Shehory, O. (2014). The evolution of mas tools. In *Agent-Oriented Software Engineering*, pages 275–288. Springer.

Švehla, D. and Rothacher, M. (2003). Kinematic and reduced-dynamic precise orbit determination of low earth orbiters. *Advances in Geosciences*, 1:47–56.

Sycara, K., Paolucci, M., Van Velsen, M., and Giampapa, J. (2003). The retsina mas infrastructure. *Autonomous agents and multi-agent systems*, 7(1):29–48.

Tabuada, P., Pappas, G. J., and Lima, P. (2001). Feasible formations of multi-agent systems. In *American Control Conference, 2001. Proceedings of the 2001*, volume 1, pages 56–61. IEEE.

Tian, Y.-P. and Liu, C.-L. (2009). Robust consensus of multi-agent systems with diverse input delays and asymmetric interconnection perturbations. *Automatica*, 45(5):1347–1353.

Tindell, K. W., Hansson, H., and Wellings, A. J. (1994). Analysing real-time communications: controller area network (can). In *1994 Proceedings Real-Time Systems Symposium*, pages 259–263.

Tomayko, J. E. (1985). Nasa's manned spacecraft computers. *Annals of the History of Computing*, 7(1):7–18.

Tortosa López, F. and Roos, P. (2005). A vhdl implementation of canopen protocol for can bus on board spacecraft. In *DASIA 2005-Data Systems in Aerospace*, volume 602.

Underwood, C., Crawford, M., and Ward, J. (1998). A low-cost modular nanosatellite based on commercial technology. In *12th AIAA/USU Conference on Small Satellites*.

Van Buijtenen, W. M., Schram, G., Babuska, R., and Verbruggen, H. B. (1998). Adaptive fuzzy control of satellite attitude by reinforcement learning. *IEEE Transactions on Fuzzy Systems*, 6(2):185–194.

Vassev, E. and Hinchey, M. (2014). Software engineering for aerospace: State of the art. In *Autonomy Requirements Engineering for Space Missions*, pages 1–45. Springer.

Viel, C., Bertrand, S., Michel, K., and Helene, P.-L. (2017). New state estimators and communication protocol for distributed event-triggered consensus of linear multi-agent systems with bounded perturbations. *IET Control Theory & Applications*.

Vinther, K., Jensen, K. F., Larsen, J. A., and Wisniewski, R. (2011). Inexpensive cubesat attitude estimation using quaternions and unscented kalman filtering. *Automatic Control in Aerospace*, 4(1).

Wahba, G. (1965). A least squares estimate of satellite attitude. *SIAM review*, 7(3):409–409.

Wang, G. and Fung, C. K. (2004). Architecture paradigms and their influences and impacts on component-based software systems. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp. IEEE.

Wang, J., Zhao, Q., and Li, H. (2014). A multi-agent based evaluation framework and its applications. *IEEE/CAA Journal of Automatica Sinica*, 1(2):218–224.

Wang, W. and McFadden, P. (1993). Early detection of gear failure by vibration analysis i. calculation of the time-frequency distribution. *Mechanical Systems and Signal Processing*, 7(3):193 – 203.

Wang, X., Liang, W., Liang, H., Zhang, A., and Wang, T. (2015). The design of spacecraft tt&c autonomous management system. In *Proceedings of the 27th Conference of Spacecraft TT&C Technology in China*, pages 491–502. Springer.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *nature*, 393(6684):440–442.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

Wen, G., Duan, Z., Yu, W., and Chen, G. (2013). Consensus of multi-agent systems with nonlinear dynamics and sampled-data information: a delayed-input approach. *International Journal of Robust and Nonlinear Control*, 23(6):602–619.

Wertz, J. R. (2012). *Spacecraft attitude determination and control*, volume 73. Springer Science & Business Media.

Whittaker, J. A. and Poore, J. H. (1993). Markov analysis of software specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2(1):93–106.

Wie, B. (2008). *Space vehicle dynamics and control*. American Institute of Aeronautics and Astronautics.

Yang, L., Gohad, T., Ghosh, P., Sinha, D., Sen, A., and Richa, A. (2005). Resource mapping and scheduling for heterogeneous network processor systems. In *Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems*, ANCS '05, pages 19–28, New York, NY, USA. ACM.

Yin, S., Ding, S. X., Haghani, A., Hao, H., and Zhang, P. (2012a). A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process. *Journal of Process Control*, 22(9):1567–1581.

Yin, S., Ding, S. X., Haghani, A., Hao, H., and Zhang, P. (2012b). A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process. *Journal of Process Control*, 22(9):1567 – 1581.

You, K., Xiao, N., and Xie, L. (2015). *Kalman Filtering with Faded Measurements*, pages 223–237. Springer London, London.

Yu, X. and Jiang, J. (2015). A survey of fault-tolerant controllers based on safety-related issues. *Annual Reviews in Control*, 39:46–57.

Yushtein, Y., Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., Olive, X., and Roveri, M. (2011). System-software co-engineering: Dependability and safety perspective. In *Space Mission Challenges for Information Technology (SMC-IT), 2011 IEEE Fourth International Conference on*, pages 18–25. IEEE.

Zagórski, P. (2012). Modeling disturbances influencing an earth-orbiting satellite. *Pomiary Automatyka Robotyka*, 16:98–103.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370.

Zeng, Z., Zhang, S., Xing, Y., and Cao, X. (2014). Robust adaptive filter for small satellite attitude estimation based on magnetometer and gyro. In *Abstract and Applied Analysis*, volume 2014. Hindawi Publishing Corporation.

Zhang, Y. and Jiang, J. (2003). Fault tolerant control system design with explicit consideration of performance degradation. *IEEE Transactions on Aerospace and Electronic Systems*, 39(3):838–848.

Zolghadri, A. (2012). Advanced model-based FDIR techniques for aerospace systems: Today challenges and opportunities. *Progress in Aerospace Sciences*, 53:18–29.

A

APPENDIX A - ORBITAL ELEMENTS

The orbit of a satellite around the Earth can be treated as a two-point mass system, which is governed by the gravitational attraction of the satellite by the Earth's mass. Newton's and Kepler's laws can be used to model the orbit of a satellite by establishing a set of orbital elements to define the size, shape and orientation of that orbit. According to Montenbruck and Gill (2012), there are six classical orbital elements that can describe the characteristics mentioned above about an orbit. These are:

- Semi-major axis a.
- Eccentricity e.
- Right Ascension of the Ascending Node (RAAN) Ω_a.
- Inclination of the orbit plane i_o.
- Argument of the perigee ω_p.
- True anomaly ν_a at epoch t_0

Figure A.1 shows the geometric properties of an ellipse used to describe orbits. The shape and the size of the orbit are defined by the semi-major axis and the eccentricity. If the eccentricity $e = 0$, it means that the orbit is circular. Whenever the variable $0 < e < 1$ the orbit is elliptical. Any orbit with $e > 1$ is hyperbolic. The eccentricity of an orbit can be calculated using the information from Figure A.1 as follows

$$e = \frac{c}{a} \tag{A.1}$$

where, c is the distance between the center and the focus point of the ellipse and a is the semi-major axis distance.

The time of perigee passage t_p is needed to determine the position of a satellite at a give moment. However, according to Wertz (2012), in many cases t_p is replaced by the

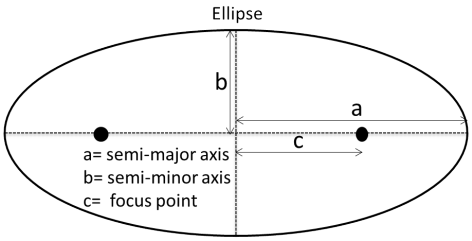


Figure A.1: Geometrical properties of a ellipse used to determine the shape of an orbit.

another parameter called Mean Anomaly M_a that measures the angle traveled by the satellite since its perigee.

The Right Ascension of the Ascending Node (RAAN) Ω_a is the angle measured from the vernal equinox counter clockwise to the ascending node, which corresponds with the point where the satellite crosses the Earth's equatorial plane (from south to north).

The inclination i_o of an orbit is the angle measured between the Earth's equatorial plane and the orbital plane. If the inclination of an orbit is zero, then it is called an equatorial orbit, while if $i_o = 90$ [deg], it is called a polar orbit. If $i_o < 90$ [deg] the orbit is called a prograde orbit, otherwise if $i > 90$ [deg], it is called retrograde.

The argument of perigee ω_p is describes the angle measured between the ascending node and the perigee following the direction of motion of the spacecraft.

The orbital elements are depicted as in the work of Jensen and Vinther (2010) and shown in Figure A.2.

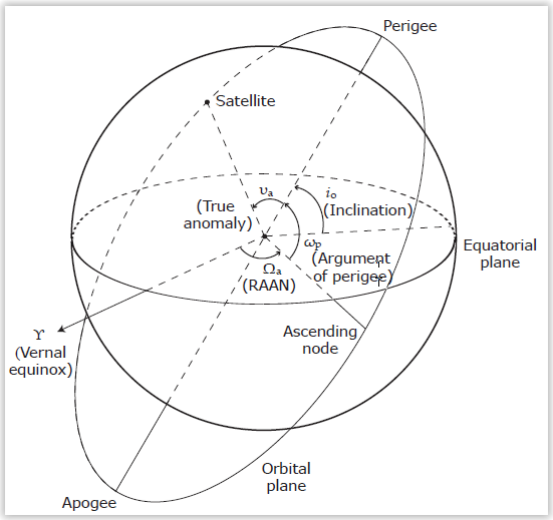


Figure A.2: Keplerian orbit around the Earth described with the modified classical orbital elements described above. Source: Jensen and Vinther (2010)

B

APPENDIX B - TWO LINE ELEMENTS

Two-Line Elements (TLE) are orbit data of man-made objects in a specific format. The data format is provided by North American Aerospace Defense Command (NORAD) for encoding a list of orbital elements for a man-made Earth-orbiting objects given a specific epoch. The TLE format consist of two lines of 69 characters each following the format presented in Figure B.1.

The TLE data representation can be used as a data source for projecting the future and past orbital tracks of space objects so that they can help to avoid collisions of space systems. There are several orbit propagation models that can use TLE as an input, for instance SGP, SGP4 and SGP8 for near-Earth objects and SDP4 and SDP8 for deep-space objects.

There are some sources for TLE that may be of interest. The majority of TLE publicly-available are distributed via Space-Trak. The data are freely available. An example of TLE for the International Space Station (ISS) is as follows:

```
ISS (ZARYA)
1 25544U 98067A   08264.51782528 -.00002182  00000-0 -11606-4 0  2927
2 25544  51.6416 247.4627 0006703 130.5360 325.0288 15.72125391563537
```

```
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNN
```

Line 1	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
08	Classification (U=Unclassified)
10-11	International Designator (Last two digits of launch year)
12-14	International Designator (Launch number of the year)
15-17	International Designator (Piece of the launch)
19-20	Epoch Year (Last two digits of year)
21-32	Epoch (Day of the year and fractional portion of the day)
34-43	First Time Derivative of the Mean Motion
45-52	Second Time Derivative of Mean Motion (decimal point assumed)
54-61	BSTAR drag term (decimal point assumed)
63	Ephemeris type
65-68	Element number
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Line 2	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
09-16	Inclination [Degrees]
18-25	Right Ascension of the Ascending Node [Degrees]
27-33	Eccentricity (decimal point assumed)
35-42	Argument of Perigee [Degrees]
44-51	Mean Anomaly [Degrees]
53-63	Mean Motion [Revs per day]
64-68	Revolution number at epoch [Revs]
69	Checksum (Modulo 10)

Figure B.1: Two-line Elements format and description of fields. Source: CelesTrak Webpage

C

APPENDIX C - EXTENDED KALMAN FILTER

The Extended Kalman Filter (EKF) is a family of non-optimal, non-linear state estimators in which the filter linearizes its dynamics about an estimate of the current mean and covariance. Depending on their attitude representation they can be divided into three categories: minimal representation EKF, Multiplicative EKF and the Additive EKF.

According to Simon (2006) and Bellantoni and Dodge (1967), the formulation of the EKF is as follows. Consider a non-linear system with discrete-time dynamics described as

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \\ \mathbf{y}_k &= h(\mathbf{x}_k) + \mathbf{v}_k \end{aligned} \tag{C.1}$$

where, $\mathbf{x}_k \in \mathbb{R}^n$ represents the state vector of the system at time k, $\mathbf{y}_k \in \mathbb{R}^m$ is the measurement vector at time k, $\mathbf{u}_k \in \mathbb{R}^l$ is the exogenous input vector, $\mathbf{w}_{k-1} \in \mathbb{R}^n$ is the process noise, and $\mathbf{v}_k \in \mathbb{R}^m$ is the measurement noise. The process noise $\mathbf{w}_{k-1}$ and the measurement noise $\mathbf{v}_k$, are assumed independent zero mean white Gaussian noises with process error covariance matrix $\mathbf{Q}_{k-1}$ and measurement noise covariance matrix $\mathbf{R}_k$, respectively.

Both, f and h are functions used to define the non-linear system dynamics. The EKF algorithm requires proper initialization of $\mathbf{x}_0$ and $\mathbf{P}_0$ and covariance matrices tuning before its implementation. After it is initialized, the Extended Kalman Filter follows two steps during each execution cycle: State Prediction and State Update.

State Prediction:

The process of estimating the current state given the previous state estimate and the observed exogenous inputs is called priori state estimate $\mathbf{x}_{k|k-1}$. It is calculated using the non-linear system model as

$$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1|k-1}, \mathbf{u}_{k-1}). \tag{C.2}$$

Numerical methods, for instance Runge Kutta (Butcher (2007)) or Euler, are used to approximate the solution of (C.2).

Also, an a priori estimate of the error covariance matrix $\mathbf{P}_{k|k-1}$ is calculated using its previous value $\mathbf{P}_{k-1|k-1}$ as

$$\mathbf{P}_{k|k-1} = \boldsymbol{\phi}_{k-1|k-1}\mathbf{P}_{k-1|k-1}\boldsymbol{\phi}^T_{k-1|k-1} + \mathbf{Q}_{k-1} \tag{C.3}$$

where, $\boldsymbol{\phi}_{k-1|k-1}$ is the Jacobian matrix for the linearized process calculated as follows

$$\boldsymbol{\phi}_{k-1|k-1} \approx \left.\frac{\partial f_{k-1}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{k-1|k-1}}. \tag{C.4}$$

State Update:

After the priori state estimate and the priori estimate of the error covariance matrix were calculated, they are used to calculate the transformed measurements propagating them through the sensor model $\mathbf{y}_{k|k-1}$ as

$$\mathbf{y}_{k|k-1} = h(\mathbf{x}_{k|k-1}). \tag{C.5}$$

The a priori state estimate $\mathbf{x}_{k|k-1}$ is updated using the measurements at time k $\mathbf{y}_k$, and the Kalman gain $\mathbf{K}$. This is called the posteriori state estimate $\mathbf{x}_{k|k}$, which is calculated as

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{y}_{k|k-1}). \tag{C.6}$$

The Kalman gain $\mathbf{K}_k$ is used to minimize the mean squared error between the predicted value for the sensors and the actual measurements. It is calculated as

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T_k(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}^T_k + \mathbf{R}_k)^{-1} \tag{C.7}$$

where, $\mathbf{H}_k$ is the Jacobian matrix for the measurement function $h(\mathbf{x}_k)$ calculated as

$$\mathbf{H}_k \approx \left.\frac{\partial h_k}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{k|k-1}}. \tag{C.8}$$

The last update to perform is the estimated error covariance matrix $\mathbf{P}_{k|k}$, which is calculated as

$$\mathbf{P}_{k|k} = (1 - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}. \tag{C.9}$$

CURRICULUM VITÆ

Johan Carvajal-Godínez was born on May 21st, 1981 in San José, Costa Rica. He attended the national system of scientific high schools (CCC) in Pérez Zeledón, where he completed his middle studies with the highest honors.

After 1999, Johan began his studies in Electronic Engineering at Costa Rica Institute of Technology until September 2004 when he graduated. From 2004 to 2007, Johan developed his professional career in the industry as a Test Process Quality Engineer at Intel, where he was leading the qualification of manufacturing factories in Costa Rica and Malaysia. In 2008, he joined as lecturer at the Costa Rica Institute of Technology in the Electronic Engineering Department. In parallel, he completed the Master on Industrial Engineering (M.Eng.) at Costa Rica Institute of Technology. In 2012, he completed the International Space Training organized by the Korean Aerospace Research Institute (KARI) in Daejon, Republic of Korea.

Between 2010 and 2011 Johan led various research initiatives at Costa Rica Institute of Technology. These activities included the development of nanotechnology research plan, an e-Science Laboratory for super-computing related research, and in 2013 when he was the project manager of the first Costa Rican satellite, he got a grant for completing his PhD on Aerospace Engineering abroad.

In 2014 Johan joined the Space Engineering Section, in the Space Engineering Department at the Faculty of Aerospace Engineering of Delft University in the Netherlands. His research focus was on onboard computer and architectures for fault-tolerant software in satellites. During his research period at Delft, he kept supporting the development activities of the Costa Rican satellite called Irazú, as a Systems Engineer. He also supported the core team of the Delfi-PQ satellite, focused on the onboard computer design. In 2018 Johan co-founded the Space Systems Laboratory of Costa Rica Institute of Technology, where he was appointed as a researcher starting on January 2019. His interests includes model-based systems engineering, onboard computer design, and fault-tolerant software architectures for guidance and navigation control of satellite systems.

LIST OF PUBLICATIONS

Journal Papers

4. Samantha Interiano-Valverde, Davide Scazzoli, Carmen Chan-Zheng, **Johan Carvajal Godinez** and Maurizio Magarini *"A Communication Protocol Design for a Multi-Agent System Framework used in Miniaturized Satellite Systems"*, Revista Tecnologia En Marcha, 33 (4) November (2020).

3. **Johan Carvajal Godinez**, Jian Guo and Eberhard Gill *"Effects of Saturation for High Throughput Satellite Buses"*, IEEE Transactions on Aerospace and Electronic Systems pages 1014 - 1025, Volume: 56 , Issue: 2 , April, (2020).

2. Carmen Chan-Zheng, **Johan Carvajal Godinez** *"A Multi-Agent System Framework for Miniaturized Satellite "*, Revista Tecnologia En Marcha, 32 (1), pp 54-67 (2019) .

1. **Johan Carvajal Godinez**, Jian Guo and Eberhard Gill *"Agent-based algorithm for fault detection and recovery of gyroscope's drift in small satellite missions"*, Acta Astronautica 139, pp 181-188 (2017).

Conference Papers

3. **Johan Carvajal Godinez**, Morteza Haghayegh, Jaan Viru, Allan Granados and Jian Guo *"Increasing computing performance of ADCS Subsystems in Small Satellites for Earth Observation"*, 10th IAA Symposium on Small Satellites for Earth Observation, Berlin, Germany 2015.

2. Stefano Speretta, Tatiana Perez-Soriano, Jasper Bouwmeester, **Johan Carvajal Godinez**, Alessandra Menicucci, Trevor Watts, Prem Sundaramoorthy, Jian Guo, and Eberhard Gill *"CubeSats to PocketQubes: Opportunities and Challenges"*, 67th International Astronautical Congress (IAC),Guadalajara, Mexico 2016.

1. **Johan Carvajal Godinez**, Jian Guo, and Eberhard Gill *"Achieving Consensus in Distributed Software Architecture for Satellite Missions"*, 69th International Astronautical Congress (IAC), Bremen, Germany, 2018.

Workshop Presentations

1. **Johan Carvajal Godinez**, Erik Orsel and Jian Guo *"CAN in Cubesats/Minisats - A Survey"*, CAN in Space Workshop (ESA/ESTEC 2016).

ACKNOWLEDGEMENTS

I want to thank all and everyone involved in my PhD process for all your support and patience. First, I want to thank the Delft University of Technology for opening me a door to come and develop my research at the Space Engineering Department. Especial thanks to Prof. Dr. Eberhard Gill and Dr. Jian Guo for their guidance and support. Your experience have been a key asset in the conclusion of this Dissertation book. Also to my fellow SSE colleagues Barry, Hans, Samiksha, Prem, Alessandra, Stefanno, Silvana, Tatiana, Nuno, Sevket. Also to thanks to Debby and Marielle for the support and help provided during these years. I want to thank my brothers and sisters in-PhD: Adolfo, Daduí, Marsil, Minghe, Dennis, Victor, Zixuan, Fiona, Ling-Yu, Mario, Salvatore, Stefano M, Lorenzo, Roberto, Fabricio, Tiago, Jin and many others that made my stay in Delft funny and pleasant.

I want to thank the Costa Rica Institute of Technology for the economical support given during this period. In particular, to the Rector Dr. Julio Calvo and Ing. Luis Paulino Mendez for their words of motivation that made me take the decision of pursuing a PhD abroad. Especially, I want to thank Adolfo Chaves Campos (q.D.g) for all the discussions and pieces of advice that made me come back to work in the academy. I also want to thank Ing. William Marín and Ing. Ara Villalobos for supporting me to get the financial means to develop this project.

This won't be nothing without my family support and sacrifice. Also, I want to thank my parents (Alexis Carvajal-Arias and Ana Godínez-Vargas), brothers (Ronald, David, Greivin, Yermy) and sister (Yorleny) for their prayers and words of support. I love you all.

There are also friends that I did not know I had until we found each other here. Thanks to the Costa Rican Crew! You made my life joyful in Delft (especially during Macumbas). Also, thanks to friends in Groningen (Mauricio, Carmen) for all your support. Ana, Victor, Tomas, Martha, Fernanda, Zoe, Carina, Jette, Isidora, Pamela Jensen, thank you for being kind to me!

I want to specially thank to Melissa Campos, who help me a lot with the translation of my summary and propositions. I am very grateful for that.

Last, but not least important I want to thank to Becki Corrales, she was key to help me get back on track to finish my PhD during 2020. Without your support this work would not be possible. I thank God from my heart that I found you in my life.

I am sorry if I forgot anyone, but you know guys and gals that you are also special, and you have touch my soul in one way or another.

Thank you all, this is yours too!