



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES  
**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

sid.inpe.br/mtc-m21c/2018/09.11.17.51-TDI

## **A REFERENCE ARCHITECTURE FOR SATELLITE SYSTEMS OPERATIONS**

Adair José Rohling

Doctorate Thesis of the Graduate Course in Space Technology and Engineering, guided by Drs. Walter Abrahão dos Santos, and Maurício Gonçalves Vieira Ferreira, approved in October 10, 2018.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34R/3RQJ3F2>>

INPE  
São José dos Campos  
2018

**PUBLISHED BY:**

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GBDIR)

Serviço de Informação e Documentação (SESID)

CEP 12.227-010

São José dos Campos - SP - Brasil

Tel.:(012) 3208-6923/7348

E-mail: pubtc@inpe.br

**COMMISSION OF BOARD OF PUBLISHING AND PRESERVATION  
OF INPE INTELLECTUAL PRODUCTION (DE/DIR-544):****Chairperson:**

Dr. Marley Cavalcante de Lima Moscati - Centro de Previsão de Tempo e Estudos Climáticos (CGCPT)

**Members:**

Dra. Carina Barros Mello - Coordenação de Laboratórios Associados (COCTE)

Dr. Alisson Dal Lago - Coordenação-Geral de Ciências Espaciais e Atmosféricas (CGCEA)

Dr. Evandro Albiach Branco - Centro de Ciência do Sistema Terrestre (COCST)

Dr. Evandro Marconi Rocco - Coordenação-Geral de Engenharia e Tecnologia Espacial (CGETE)

Dr. Hermann Johann Heinrich Kux - Coordenação-Geral de Observação da Terra (CGOBT)

Dra. Ieda Del Arco Sanches - Conselho de Pós-Graduação - (CPG)

Silvia Castro Marcelino - Serviço de Informação e Documentação (SESID)

**DIGITAL LIBRARY:**

Dr. Gerald Jean Francis Banon

Clayton Martins Pereira - Serviço de Informação e Documentação (SESID)

**DOCUMENT REVIEW:**

Simone Angélica Del Ducca Barbedo - Serviço de Informação e Documentação (SESID)

André Luis Dias Fernandes - Serviço de Informação e Documentação (SESID)

**ELECTRONIC EDITING:**

Marcelo de Castro Pazos - Serviço de Informação e Documentação (SESID)

Murilo Luiz Silva Gino - Serviço de Informação e Documentação (SESID)



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES  
**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

sid.inpe.br/mtc-m21c/2018/09.11.17.51-TDI

## **A REFERENCE ARCHITECTURE FOR SATELLITE SYSTEMS OPERATIONS**

Adair José Rohling

Doctorate Thesis of the Graduate Course in Space Technology and Engineering, guided by Drs. Walter Abrahão dos Santos, and Maurício Gonçalves Vieira Ferreira, approved in October 10, 2018.

URL of the original document:

<<http://urlib.net/8JMKD3MGP3W34R/3RQJ3F2>>

INPE  
São José dos Campos  
2018

Cataloging in Publication Data

---

Rohling, Adair José.

R636r      A reference architecture for satellite systems  
operations / Adair José Rohling. – São José dos Campos :  
INPE, 2018.

xx + 122 p. ; (sid.inpe.br/mtc-m21c/2018/09.11.17.51-TDI)

Thesis (Doctorate in Space Technology and Engineering) –  
Instituto Nacional de Pesquisas Espaciais, São José dos Campos,  
2018.

Guiding : Drs. Walter Abrahão dos Santos, and Maurício  
Gonçalves Vieira Ferreira.

1. Satellites control. 2. Reference architecture. 3. Software  
component. I.Title.

CDU 629.783:004.4

---



Esta obra foi licenciada sob uma Licença [Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada](#).

This work is licensed under a [Creative Commons Attribution-NonCommercial 3.0 Unported License](#).

Aluno (a): **Adair José Rohling**

Título: "A REFERENCE ARCHITECTURE FOR SATELLITE SYSTEMS OPERATIONS"

Aprovado (a) pela Banca Examinadora  
em cumprimento ao requisito exigido para  
obtenção do Título de **Doutor(a)** em

**Engenharia e Tecnologia Espaciais/Eng.  
Gerenc. de Sistemas Espaciais**

Dr. Nilson Sant'Anna

  
\_\_\_\_\_  
Presidente / INPE / SJC Campos - SP

( ) Participação por Vídeo - Conferência

☒ Aprovado ( ) Reprovado

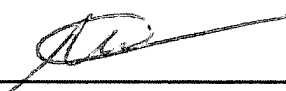
Dr. Walter Abrahão dos Santos

  
\_\_\_\_\_  
Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Vídeo - Conferência

☒ Aprovado ( ) Reprovado

Dr. Maurício Gonçalves Vieira Ferreira

  
\_\_\_\_\_  
Orientador(a) / INPE / SJC Campos - SP

( ) Participação por Vídeo - Conferência

☒ Aprovado ( ) Reprovado


Dra. Elisa Yumi Nakagawa

  
\_\_\_\_\_  
Convidado(a) / USP / São Carlos - SP

☒ Participação por Vídeo - Conferência

☒ Aprovado ( ) Reprovado

Dr. Valdemar Vicente Graciano Neto

  
\_\_\_\_\_  
Convidado(a) / UFG / Goiânia - GO

☒ Participação por Vídeo - Conferência

☒ Aprovado ( ) Reprovado

Este trabalho foi aprovado por:

( ) maioria simples

☒ unanimidade

São José dos Campos, 10 de outubro de 2018



## ACKNOWLEDGEMENTS

(In Portuguese)

A Deus pela minha vida e saúde.

Agradeço especialmente minha família, minha mãe e meu pai pela vida e educação. Também às minhas irmãs e especialmente minha esposa e filhas Ana Clara, Maria e Dayana, que apesar das dificuldades sempre me fortaleceram.

Aos Drs. Mauricio Gonçalves Vieira Ferreira e Walter Abrahão dos Santos: obrigado por todas as orientações, incentivos e pela confiança depositada. À professora Eliza Yumi Nakagawa e ao amigo Valdemar Vicente Graciano Neto pelos ensinamentos recebidos durante o período de doutorado sanduíche na USP.

Agradeço ao INPE, UTFPR e ao CNPq. E também a todas as pessoas que se fizeram presentes e contribuíram na realização deste trabalho.





## ABSTRACT

Software for Satellite Control Systems (SCS) domain performs a relevant role in space systems, being responsible for ensuring the functioning of the satellites, from the orbit launch to the end of their lifetime. Systems in this domain are complex and are constantly evolving due to technological advancement of satellites, the significant increase of controlled satellites, and the interoperability among space organizations. However, in order to meet such complexity and such evolution, the architectures of these systems have been usually designed in an isolated way by each organization, hence may be prone to recurrent efforts and difficulties of interoperability. In parallel to this scenario, reference architecture, a special type of software architecture that aggregates knowledge of a specific domain, has performed an important role for the success in development, standardization, and evolution of systems in several domains. Nevertheless, the usage of reference architecture has not been explored in the SCS domain. Thus, this thesis presents a Reference Architecture for Satellite Operations Systems. Results achieved from using this reference architecture in the development of a Microsatellite Control System for National Institute for Space Research (INPE) showed a significant reduction of effort, benefits of interoperability, scalability, and sharing of ground resources.

Keywords: Satellites Control. Reference Architecture. Software Component.



# UMA ARQUITETURA DE REFERÊNCIA PARA SISTEMAS DE CONTROLE DE SATÉLITES

## RESUMO

Software para o domínio de Sistemas de Controle de Satélites(SCS) desempenham um papel relevante em sistemas espaciais, sendo responsável por assegurar o funcionamento dos satélites, desde seu lançamento em órbita até o final de sua vida útil. Sistemas deste domínio são complexos e permanecem em constante evolução em consequência dos avanços tecnológicos dos satélites, do aumento significativo de satélites controlados e da interoperabilidade entre as organizações espaciais. No entanto, para atender essas complexidade e evoluções, as arquiteturas desses sistemas geralmente são projetadas de forma isolada em cada organização, e assim podem estar propensas a esforços recorrentes e dificuldades de interoperabilidade. Paralelamente a este cenário, arquitetura de referência, um tipo especial de arquitetura de software que agrega conhecimento de um domínio específico, desempenha um papel importante para o sucesso no desenvolvimento, padronização e evolução de sistemas em vários domínios. No entanto, o uso de arquitetura de referência não tem sido explorada em sua completude no domínio de SCS. Assim, o objetivo deste trabalho é estabelecer uma Arquitetura de Referência para Sistemas de Controle de Satélites. Resultados obtidos com o uso desta arquitetura de referência no desenvolvimento de um Sistema de Controle de Microssatélites para o Instituto Nacional de Pesquisas Espaciais (INPE) apresentou uma redução significativa de esforços, benefícios de interoperabilidade, escalabilidade e compartilhamento de recursos em solo. Como consequência, custos geralmente gastos no desenvolvimento e evolução de SCS podem ser reduzidos.

Palavras-chave: Controle de Satélites. Arquitetura de Referência. Componentes de Software.



## LIST OF FIGURES

	<u>Page</u>
2.1 CubeSats by Mission Type . . . . .	10
2.2 Small satellites Launch History and Forecast (1-50 kg). . . . .	13
2.3 Small satellites trends by sector (1-50 kg). . . . .	14
2.4 Sequence of the Control Operations. . . . .	16
2.5 Illustration of data flow style . . . . .	19
2.6 The structure of a typical web-based application . . . . .	19
2.7 Client-server structure . . . . .	20
2.8 Structure pattern of call and return architecture . . . . .	21
2.9 Structure pattern of layered systems . . . . .	22
2.10 Summary of software architectural styles and sub-styles. . . . .	23
2.11 Diagram UML Version 2.5. . . . .	26
2.12 Archimate - Layers and Aspects . . . . .	27
2.13 Relationship between reference architecture and product line architecture. . . . .	29
2.14 Relationship between reference model, reference architecture, concrete software architectures and software application. . . . .	30
2.15 Outline structure of ProSA-RA. . . . .	31
2.16 Groups and Elements of RAModel. . . . .	33
2.17 Component lifecycle . . . . .	34
2.18 Architectural Specialization . . . . .	36
2.19 CORBA Component Model . . . . .	37
2.20 Process Model of CBD . . . . .	39
 3.1 High-level Decomposition of the EGS-CC. . . . .	 43
3.2 Decomposition of the EGS-CC Kernel . . . . .	44
3.3 Overview of technology suite - EGC-CC. . . . .	45
3.4 CCT Execution Architecture: Data Flow View. . . . .	46
3.5 CCT Planning Architecture: Data Flow View. . . . .	47
3.6 Hifly SOA Layer. . . . .	48
3.7 Hifly anywhere SOAP services logical model. . . . .	49
3.8 Overview of Hifly components and interfaces with GMV products. . . . .	50
3.9 SCOS-2000 client/server architecture. . . . .	51
3.10 GMSEC Layered Architecture. . . . .	54
3.11 Screen Shot of the GEDAT. . . . .	55
3.12 GEDAT - resource information (CPU, memory, network, disk utilization). . . . .	55
3.13 Architecture of the GENSO network. . . . .	57

3.14	Genso layered architecture. . . . .	58
3.15	SatNet architecture overview. . . . .	59
3.16	SatNet integration. . . . .	62
4.1	Enterprise View of SCS-RA in organizational structures ECSS Standard.	73
4.2	Structural View of SCS-RA. . . . .	74
4.3	Data Flow View. . . . .	75
4.4	Logical View of SCS-RA. . . . .	76
4.5	Composition View of SCS-RA. . . . .	76
5.1	MicroSatCS Development Life Cycle. . . . .	86
5.2	Logic View of MicroSatCS. . . . .	87
5.3	Deployment View of MicroSatCS. . . . .	88
5.4	CubeSoft Taxonomy . . . . .	89
5.5	Instantiation of MicroSatCS. . . . .	91
5.6	MicroSatCS and Tracking of Satellite. . . . .	92
5.7	Computational Resources - Gateway Component . . . . .	92
5.8	Monitoring Component - Tancredo-I picosat telemetry in the UbatubaSat Project. . . . .	93

## LIST OF TABLES

	<u>Page</u>
2.1 Classification of satellites by mass . . . . .	9
2.2 Small Satellites by Field of Knowledge . . . . .	12
2.3 Small satellites - Brazil . . . . .	14
2.4 Component Models and Purpose . . . . .	35
3.1 CCT-Subsystems, Categories and Components . . . . .	46
4.1 Contents of interest captured for the definition of SCS-RA . . . . .	65
4.2 Mapping of the Features in Existing Systems . . . . .	66
4.3 SCS-RA Requirements . . . . .	66
4.4 Category . . . . .	69
4.5 Requirements Mapping to the Architectural Design . . . . .	77
4.6 Mapping of the SCS-RA components to the Existing Systems . . . . .	80
4.7 Comparison between the elements of RAModel and SCS-RA. . . . .	81





## LIST OF ABBREVIATIONS

ADL	– Architectural Description Language
ATAM	– Architecture Tradeoff Analysis Method
CAT	– Criteria Action Table
CBD	– Component Based Development
CCM	– CORBA Component Model
CORBA	– Common Object Request Bro-ker Architecture
CCSDS	– Consultative Committee for Space Data Systems
CCT	– Control Channel Toolkit
COTS	– Commercial off-the-shelf
EGS-CC	– European Ground Systems – Common Core
EJB	– Enterprise Java Beans
ESOC	– European Space Operations Center
FERA	– Framework for Evaluation of Reference Architectures
GEDAT	– GMSEC Environment Diagnostic Analysis Tool
GMSEC	– The Goddard Mission Services Evolution Center
GRASP	– GMSEC Remote Access Service Provider
GREAT	– GMSEC Reusable Event Analysis Toolkit
GSFC	– Goddard Space Flight Center
HTTPS	– Hypertext Transfer Protocol Secure
MOM	– Message Oriented Middleware
NRO	– United States National Reconnaissance Office
OSGi	– Open Services Gateway Initiative
RAModel	– Reference Architecture Model
RASIM	– Reference Architecture for Space Information Management
RASDS	– Reference Architecture for Space Data Systems
RPC	– Remote Procedure Calls
SA	– System Agent
SAAM	– Software Architecture Analysis Method
SCS	– Satellites Control System
SCOS 2000	– Satellite Control and Operation System 2000
SEI	– Software Engineering Institute
SOA	– Service Oriented Architecture
SOAP	– Simple Object Access Protocol



# CONTENTS

	<u>Page</u>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Justification for the Research	2
1.2 Objectives	4
1.3 Thesis Structure	4
<b>2 BACKGROUND</b>	<b>7</b>
2.1 Space System	7
2.2 Satellites	7
2.2.1 Historical Context	8
2.2.2 Classification	9
2.2.3 CubeSat	9
2.2.4 Applications for Small Satellites	11
2.2.5 Prospects for Small Satellites	12
2.2.6 Small Satellites in Brazil	13
2.3 Satellite Control	13
2.3.1 Satellite Control Center	15
2.3.1.1 INPE's Satellites Control Center	17
2.4 Software Architecture	17
2.4.1 Architectural Style	18
2.4.1.1 Data Flow Architectural Style	19
2.4.1.2 Independent Component Architectural Style	20
2.4.1.3 Call and Return Architectural Style	21
2.4.1.4 Data-Center Architectural Style	22
2.4.1.5 Virtual Machine Architectural Style	22
2.4.2 Architectural View	22
2.4.3 Architectural Representation	25
2.5 Reference Architecture	27
2.5.1 ProSA-RA: A Process to Build Reference Architectures	31
2.5.1.1 Step RA-1: Information Source Investigation	31
2.5.1.2 Step RA-2: Architectural Analysis	32
2.5.1.3 Step RA-3: Architectural Synthesis	32
2.5.1.4 Step RA-4: Architectural Evaluation	32

2.6	Software Components . . . . .	32
2.6.1	Components Model . . . . .	34
2.6.1.1	Enterprise Java Beans (EJB) . . . . .	36
2.6.1.2	CORBA Component Model (CCM) . . . . .	36
2.6.1.3	Open Services Gateway Initiative (OSGi) . . . . .	37
2.7	Component-Based Development . . . . .	38
2.7.1	Component-Based Development Process . . . . .	39
<b>3</b>	<b>RELATED WORK . . . . .</b>	<b>41</b>
3.1	Reference Architecture for Space Systems . . . . .	41
3.2	Software Application for Satellites Control Systems . . . . .	42
3.2.1	The European Ground Systems – Common Core (EGS-CC) . . . . .	42
3.2.2	Control Channel Toolkit (CCT) . . . . .	44
3.2.3	Hifly . . . . .	47
3.2.4	Satellite Control and Operation System 2000 (SCOS-2000) . . . . .	51
3.2.5	Goddard Mission Services Evolution Center (GMSEC) . . . . .	52
3.2.6	Global Educational Network for Satellite Operations (Genso) . . . . .	56
3.2.7	Satellite Network (SatNet) . . . . .	58
3.2.7.1	SatNet Architecture . . . . .	58
3.2.7.2	SatNet Services . . . . .	60
3.2.7.3	SatNet Integration . . . . .	61
<b>4</b>	<b>ESTABLISHMENT OF A REFERENCE ARCHITECTURE FOR SATELLITES CONTROL SYSTEM . . . . .</b>	<b>63</b>
4.1	Step RA-1: Information Source Investigation . . . . .	63
4.2	Step RA-2: Architectural Analysis . . . . .	66
4.2.1	Architectural Requirements of SCS Domain . . . . .	66
4.2.2	Components of SCS Domain . . . . .	68
4.2.2.1	Registry Component . . . . .	69
4.2.2.2	Mashup Component . . . . .	69
4.2.2.3	Discovery Component . . . . .	70
4.2.2.4	Composition Component . . . . .	70
4.2.2.5	Orbit Calculator Component . . . . .	70
4.2.2.6	Telemetry Component . . . . .	71
4.2.2.7	Maneuver Component . . . . .	71
4.2.2.8	Telecommand Component . . . . .	71
4.2.2.9	Schedule Component . . . . .	71
4.2.2.10	Spacecraft Component . . . . .	71

4.2.2.11	Ground Station Component . . . . .	72
4.2.2.12	Tracking Component . . . . .	72
4.2.2.13	Gateway Component . . . . .	72
4.3	Step RA-3: Architectural Synthesis . . . . .	72
4.3.1	Enterprise View . . . . .	72
4.3.2	Structural View . . . . .	73
4.3.3	Data Flow View . . . . .	74
4.3.4	Logical View . . . . .	74
4.3.5	Composition View . . . . .	75
4.4	Step RA-4: Architectural Evaluation . . . . .	77
4.4.1	Requirements Mapping to the Architectural Design . . . . .	77
4.4.2	Validation of the SCS-RA by Mapping of the SCS-RA Components to the Existing Systems . . . . .	80
4.4.3	Validation through the RAModel . . . . .	81
4.5	Final Remarks . . . . .	84
<b>5</b>	<b>CASE STUDY - USING SCS-RA FOR DEVELOPMENT OF MICROSATELLITES CONTROL SYSTEM . . . . .</b>	<b>85</b>
5.1	MicroSatCS Overview . . . . .	85
5.2	The MicroSatCS Development Process . . . . .	85
5.3	Architectural Design Process . . . . .	87
5.4	Software Design & Implementation Process . . . . .	88
5.5	Software Operation Process . . . . .	91
5.6	Final Remarks . . . . .	93
<b>6</b>	<b>CONCLUSIONS AND FUTURE WORKS . . . . .</b>	<b>95</b>
6.1	Discussion . . . . .	95
6.2	Final Remarks . . . . .	95
6.3	List of Publications Attained . . . . .	96
	<b>REFERENCES . . . . .</b>	<b>101</b>



## 1 INTRODUCTION

Satellites currently comprise an essential technology with high impact to society. They support and offer important services, such as telecommunication, global positioning system (GPS), weather forecast, earth and space observation, meteorology, resource monitoring, military observation, and many others. INPE, as one of the main Brazilian organization for space technology, has concentrated its efforts to promote studies and scientific research for the technological and space environment, since its creation in 1961. There are numerous projects being developed and executed in the space area, including satellites SCD1, SCD2, SACS1, SACS2, SATEC, CBERS-1, CBERS-2, CBERS-2B, CBERS-3, CBERS-4 e CBERS-4A, Amazonia-1 and EQUARS. INPE has a Satellite Control Center (SCC) responsible for the planning and execution of satellite control operations. The SCC has developed software solutions for Satellites Control Systems (SCS)<sup>1</sup> since 1980s is always proposing advances and technological innovations to contribute in this domain.

In order to keep them properly functioning, satellites are controlled during their whole lifetime by SCS. These systems control satellites in orbit by performing operations developed on the ground. Software solutions for SCS are complex and have several functions, among which we can highlight: (i) sending telecommands; (ii) telemetry reception; (iii) mission data reception; (iv) satellites tracking; (v) attitude control; (vi) maneuver calculation; (vii) orbit propagation and determination; and (viii) flight plans generation. In addition to their various functionalities, such solutions must have scalability to meet the significant increase in the number of controlled satellites. According to United Nations Office for Outer Space Affairs (UNOOSA) ([UNITED NATIONS OFFICE FOR OUTER SPACE AFFAIRS \(UNOOSA\), 2017](#)), 7,853 objects were launched into outer space, excluding space debris and non-functional objects. As reported to Union of Concerned Scientists (UCS) ([UNION OF CONCERNED SCIENTISTS \(UCS\), 2017](#)), there were 1,459 operational satellites by the end of 2016. The Nano / Microsatellite market forecast ([SPACEWORKS ENTERPRISES, 2017](#)) shows a 70% increase in the number of launches of small satellites (up to 50 kg) from 2017 to 2023. In Brazil, for example, there is also a significant increase in the number of projects and initiatives for the development and launch of small satellites, such as NANOSATC-BR1, NANOSATC-BR2, AESP, CONASAT, ITASAT, EQUARS, UbatubaSat, SERPENS and FLORIPA-SAT.

Space organizations have several ground-based infrastructure resources to control

---

<sup>1</sup>For sake of simplicity, henceforth, we use this acronym to express both singular and plural.

satellites. Examples of these resources are antennas, rotors, modems, radio, Terminal Node Controller (TNC), computers, softwares, and data links. Most of them are only used during the satellites passages over their ground stations. The sharing of these resources among space organizations increases the number of accesses to the satellites, provides greater capacity to control and track them, and significantly reduces entire cost of a space mission (WERTZ; LARSON, 1999). The sharing of ground resources is addressed by the Consultative Committee for Space Data Systems (CCSDS) through the standardization of Cross Support Services(CSS) (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2006). The cross support occurs when one organization provides part of its resources to service the space data transfer requirements of another organization (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2005). On the ground segment, CSS are defined through Space Link Extension (SLE) services. In recent years, European Space Agency (ESA) has adopted the CCSDS Recommendations for SLE transfer services as a standard interface for network ESTRACK (ARZA; DREIHAHN, 2012). The NASA/JPL also adopted the CCSDS Recommendations SLE for using in Deep Space Network(DSN) (NASA, 2014; NASA, 2015). The sharing of resources in ground is fundamental and has several advantages, so that when we develop softwares for SCS, they can meet interoperability requirements among space organizations to make possible the sharing of ground resources.

In short, Software systems for SCS are complex and remain in constant evolution to cope with: (i) the diversity and complexity of space missions (as a consequence of the advances in electronics and the increase of computational power, which enable satellites to perform diverse and extremely complex tasks in the space) (DVORAK, 2009; SHAMES; YAMADA, 2003); (ii) the significant increase in the number of controlled satellites, mainly small ones (SORENSEN et al., 2012; SPACEWORKS ENTERPRISES, 2017); and (iii) the interoperability and sharing of ground resources among space organizations (SORENSEN et al., 2012; SHAMES; YAMADA, 2003; THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2005; THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2006; SHAMES; YAMADA, 2004).

## 1.1 Problem Statement and Justification for the Research

Several software applications have been developed, both in academia and in the industry, to meet the evolution and complex scenario of SCS. However, during development and upgrades of such applications, each organization designs their software architecture in an isolated way and without an architectural model, causing signif-



icant and recurrent rework efforts. These software architectures may also further require adaptations during the process of interoperability among organizations that adopt different SCS (CHAMOUN *et al.*, 2006; SULLIVAN *et al.*, 2009). Besides that, new adaptations may be necessary when performing new interoperability processes, interfering directly or indirectly with the existing ones. In Shames and Yamada (2003), Shames and Yamada (2004), THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS (2013), it is stated that elements of a given SCS cannot be easily used by others and it is also sometimes difficult even to describe the problems associated with interoperability among SCS. In this sense, there are several factors which should be considered when developing interoperable SCS solutions, e.g., high security, availability, dependability, heterogeneity, and low cost (SMITH *et al.*, 2008). However, the lack of any type of architectural model or standard services has led to non-interoperable systems (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2013). As SCS becomes more complex, having architectural models that can be used as a guideline to develop SCS architectures seem to be a suitable solution.

In this context, Reference Architectures(RA)<sup>2</sup> arise, encompassing the knowledge about how to design concrete architectures of systems by (i) capturing the essence of software architectures of a particular application domain (OUSSALAH, 2014; MARTÍNEZ-FERNÁNDEZ, 2013), (ii) serving as standardization and evolution of software systems of a given domain (NAKAGAWA *et al.*, 2011); (iii) avoiding the reinvention or revalidation of solutions to problems already solved (CLOUTIER *et al.*, 2010); and (iv) reducing costs of maintenance and development of software applications (JONES, 1986; TRACZ, 1988; WENTZEL, 1994). Due to their advantages, RA has been proposed and also successfully used in several domains, including in the industry (MARTÍNEZ-FERNÁNDEZ *et al.*, 2015; CLEMENT *et al.*, 2017; FERRO *et al.*, 2015; KLEIN *et al.*, 2016; PANUNZIO, 2011; NAKAGAWA *et al.*, 2011; GUESSI *et al.*, 2015; WEYRICH; EBERT, 2016; FILHO; BARBOSA, 2015; SHETH *et al.*, 2010). Some important initiatives have encouraged the creation of RA in space systems (SHAMES; YAMADA, 2003; THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2013; THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008; PANUNZIO; VARDANEGA, 2013; DURO *et al.*, 2005). Nevertheless, the use of RA has not been explored in the development of systems for the SCS domain.

---

<sup>2</sup>For sake of simplicity, this acronym will be used interchangeably to express singular and plural forms.

## 1.2 Objectives

According to the research gap characterized in the previous section, the general research question to be investigated in this thesis is whether or not the systematization of the reference architecture can positively impact on the quality of SCS software architectures.

Motivated by this scenario, the main goal of this work is to establish a Reference Architecture for Satellite Systems Operations. This reference architecture will be defined in the sequence as Reference Architecture for Satellite Control Systems (SCS-RA). SCS-RA has the following goals: (i) supporting the development and evolution of SCS; (ii) contributing to improve interoperability, structuring and maintaining SCS; and (iii) contributing to improve reuse of software components when developed based on SCS-RA. In order to establish SCS-RA, ProSA-RA ([NAKAGAWA et al., 2014](#)) was used as a systematized process to design, represent, and evaluate RA. The validation of SCS-RA was carried out during the development of a Microsatellites Control System (MicroSatCS). Results achieved through the adoption of SCS-RA brought evidence of its feasibility and other advantages of its applicability. An additional contribution of this research is the definition of a Software Component Model for Satellite Control Domain (CubeSoft).

## 1.3 Thesis Structure

This thesis report is comprised of 6 chapters structured as described hereafter.

An overview of the background information that supports the topics investigated in this thesis are explored in Chapter 2. Initially, key concepts related to Space System and Satellite Control are discussed. After that, theory associated with Software Architecture, Software Components, and Reference Architecture are addressed.

Chapter 3 comprises the related works and introduces some important initiatives which encourage the creation of RA in space systems. It explores some software application for Satellites Control Systems and their architectures. The European Ground Systems – Common Core (EGS-CC), Control Channel Toolkit (CCT), Hifly system, Satellite Control and Operation System 2000 (SCOS-2000), Goddard Mission Services Evolution Center (GMSEC), Global Educational Network for Satellite Operations (Genso), and Satellite Network (SatNet) are parts of this study.

Chapter 4 describes the establishment of Reference Architecture for SCS (SCS-RA). We adopted the systematic process ProSA-RA to establish the SCS-RA. The estab-

lishment of SCS-RA involved the steps of ProSA-RA: Information Source Investigation, Architectural Analysis, Architectural Synthesis, and Architectural Evaluation. The evaluation of SCS-RA was accomplished through the requirements mapping to the architectural design, mapping of the SCS-RA components to the existing systems, and through the use RAModel. This chapter too shows final remarks.

Chapter 5 shows a case study of the use of SCS-RA in the development of a MicroSatellites Control System (MicroSatCS). The development of MicroSatCS contributed to observe benefits of adopting SCS-RA and to supply evidence on its viability.

Finally, chapter 6 presents the conclusions of this study, revisiting the achieved contributions, summarizing limitations, and brings the final remarks and future work perspectives. The list of publications attained from this work is presented.



## 2 BACKGROUND

This chapter provides an overview of the subjects that underlie the research developed in this thesis. The organization of the chapter is as follows. Section 2.1 introduces the Space System. Section 2.2 presents key concepts about satellites. Section 2.3 shows an overview of Satellite Control. Sections 2.4, 2.5, and 2.6 characterizes the state-of-the-art of Software Architecture, Reference Architecture, and Software Components, respectively.

### 2.1 Space System

A space system is composed of all necessary elements for the development of a product which, at any given moment of its life cycle, should be in contact with the space environment, whether it orbits the earth as any other celestial body, any other celestial body or as it travels through space. Satellites, rockets and probes are the most visible of these products, projected in a space system.

The space system is better understood in its development through the constitution of physical structural divisions based on peculiar functions. These divisions are denominated segments, according to (PISACANE, 2005). A segment is a set of elements which is grouped to form a larger component or main function within a system. The main sections of a space system are:

- a) Space segment: this is the part placed into orbit, denominated as satellites, probes, space stations;
- b) Launch segment: it is the part used to place the space section into orbit, denominated rockets, spaceships, etc. ;
- c) Ground segment: it is the part charged with supervision of the satellite functioning. It allows controlling the satellite during its life cycle, data control and reception. It is constituted mainly by ground stations, data communication networks and the Satellite Control Center (SCC).

From now on, we will present details on satellites, which constitute the main element of the space segment.

### 2.2 Satellites

The artificial satellite is a vehicle developed by man and placed in an orbit with the aim of scientific investigation. Satellites are conceptually divided in two parts:

payload and the platform. The payload is constituted of specialized instruments destined to accomplish the mission proposed for the flight. This way, a remote sensing satellite's payload is typically a camera or a set of cameras. The platform is constituted by the necessary parts to support the launching and to serve the payload's operations.

Each satellite performs functions in a space mission, which may be telecommunications, remote sensing, data collection and scientific and technological testing ([EUROPEAN COOPERATION FOR SPACE STANDARDIZATION \(ECSS\), 2000](#); [WERTZ; LARSON, 1999](#)).

Satellites are equipped with devices to send and receive data. To keep adequate functioning, satellites are controlled by ground stations, which send instructions and receive data from the mission and telemetry data from the satellite. This way, the ground stations perform a fundamental role in the communication with the orbiting satellites.

Today, we may verify a constant reduction in the size of satellites concurrently with an increase in the operational capability. The electronic miniaturization and the increase in computational power resulted, resulted in an enormous increase in number and importance of small satellites in the last years.

The new generations of small satellites are creating significant opportunities for research and industry. Some missions which were performed by large-sized satellites before, now may be performed by small satellites ([COUNCIL, 1994](#)).

In the small satellites there is an important characteristic: the standardization of variables such as mass, form and dimensions. Because of that, the time for development and the construction costs may be significantly reduced ([PUIG-SUARI et al., 2001](#)).

### **2.2.1 Historical Context**

The origin of satellites goes back to the development of small satellites, since 1957, when the former USSR launched the first satellite to space, the Sputnik I, weighting 83.6kg. Other small satellites also inhabited space, as did the Explorer and the Vanguard, launched by the USA, weighting 8.3kg and 1.47kg, respectively. During that time, the limited capability of launchers set the restrictions for the mass and the dimensions of the satellites. The Sputnik, the Explorer and the Vanguard programs had a high success level and opened the doors to the space race. They were also

the forerunners for technologies as space thermal control, telemetry transmission and reception in orbit, and solar power and transistors, which contributed to the reduction in the size of satellites (HELVAJIAN; JANSON, 2008).

### 2.2.2 Classification

There are many different ways to classify satellites. The main classification systems are performed through characteristics as type of orbit, cost and mass (DUBOS et al., 2010; FORTESCUE et al., 2003; BOLEA-ALAMANAC, 2001; SHIROMA et al., 2011)). The most popular way to classify small satellites is the categorization by mass. Table 2.1 shows the classification of satellites by mass.

Table 2.1 - Classification of satellites by mass

Category	Class	Mass range (kg)
Standard Satellites	Large	>1000
	Medium	500-1,000
Small Satellites	Mini	100-500
	Micro	10-100
	Nano	1.0-10
	Pico	0.1-1.0
	Femto	<0.1

SOURCE: Fortescue et al. (2003)

In this class, we can find different values, as in the report by the Ames Research Center in (NASA, 2015), where the small satellites with the mini-satellites category are limited to 180kg.

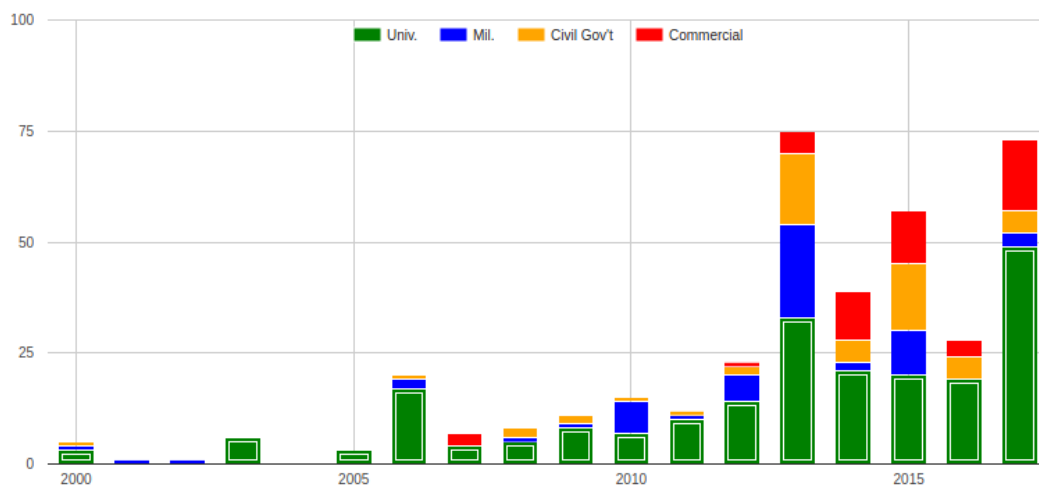
### 2.2.3 CubeSat

In 1999, the professors Jordi Puig-Suari, from California Polytechnic State (CalPoly), and Robert Twiggs, from Stanford University's Space Systems Development Laboratory (SSDL), made the document available 'Cubesat Design Specification' to standardize projects on the CubeSat standard. The document defines the requirements, physical, mechanical and electric specifications to standardize satellites to the CubeSat standard (CALPOLY, 2014). The CubeSat standard main characteristics are: its structure, 10cm in the shape of a cube; its mass, limited to 1.3kg. As a result of these efforts, smallsats can now be developed with relatively modest resources: a small team of developers, off-the-shelf components (COTS), and a

budget that is feasible and affordable for smaller organizations.

The main goal aimed by the professors by creating the CubeSat standard was to make possible for students to design, build, test and operate a satellite with similar capabilities as the first satellites developed (CHIN et al., 2008a; PUIG-SUARI et al., 2001). The CubeSat standard is present in more than 100 private and public universities and space agencies (CALPOLY, 2014). The figure 2.1 demonstrates the CubeSat’s participation in university missions as well as other mission types.

Figure 2.1 - CubeSats by Mission Type



SOURCE: Swartwout (2017)

With CubeSat recent popularization, the interest in commercial applications was been considerably expanded, especially as a consequence of the maturity of existing university technologies and projects. The CubeSat standard is helping to accelerate the development of technologies and is also making the commercial application of the platform more attractive (LOWE; MACDONALD, 2014).

The CubeSat standard success is attributed to its standardized interface in the integration of the launching vehicle, which resulted in lower launching costs, of around many tens of thousands of Euro and an accelerated schedule in the preparation for launching (SCHOLZ; JUANG, 2015).

CubeSat launchings have been done as secondary payload in different launching vehi-



cles (CHIN et al., 2008b). Shared launching between payloads is increasingly common, being good for all parties involved. The costs for producing and launching a CubeSat amount to approximately 65 to 80 thousand dollars (DAVID, 2004), and the approximate cost for launching averages 20 thousand dollars by kilogram (SUTHERLAND, 2012; LOWE; MACDONALD, 2014).

#### 2.2.4 Applications for Small Satellites

The micro-electronics technologies give small satellites the possibility to provide services in the main areas of the space section. Areas such as telecommunications, geosciences, space science and teaching have small satellites as a resource to reach the mission goals in the space section.

In the telecommunications area, there are some examples as the Brazilian satellites, SCD1 and SCD2, which provide transmission services for environmental data. There are also the American Teledesic satellites used for transmission of broadband Internet and the Globalstar satellite used to provide with telephone service to remote areas.

In the geosciences area, the small satellites have many applications, as in earth, atmosphere and ocean surveillance services. Miniaturization of technologies as electronic cameras, optical telescopes and sensors were the main factors to contribute to this progress.

The small satellites have also been used in space science, accomplishing useful functions especially in the areas of astronomy and astrophysics, such as the observation of the celestial electromagnetic spectrum. In the educational area, the small satellite enabled the students to practice the development of space systems (HEIDT et al., 2000; BOLEA-ALAMANAC, 2001; THYAGARAJAN et al., 2005). Among the small satellites with mass lower than 10kg and built until 2010, around 52% had educational purposes (BOUWMEESTER; GUO, 2010). Table 2.2 presents some of the principal projects for small satellites classified by field of knowledge.

There are other uses for small satellites, as in the use of experimental technology in the space environment, due to the cost and to the fast development of components. In military activity, the small satellites can contribute to defense systems and also in the demonstration of technology, signals intelligence, communication and observations.

Table 2.2 - Small Satellites by Field of Knowledge

Field of Knowledge	Satellites (Country)
Telecommunications	SCD1 and 2 (Brazil); UoSat (United Kingdom); Healthsat2, FAISAT, TUBSAT, ORBCOMM, Globalstar, Teledesic (United States).
Geosciences	IMAGE, OrbView-2,4, GRACE (United States); Orsted (Denmark); SMOS (France); FASat-bravo (Chile); TMSat (Thailand); KITSAT-3 (South Korea); Tsinghua (United Kingdom).
Space Science	Hiten, Lunar-A, Yohkoh, Asuka (Japan); Clementine, Mars Pathfinder, Lunar Prospector, Europa Orbiter, Deep Space1, Stardust, SWAS, TRACE, WIRE, GALEX, STEREO (United States); ABRIXAS, DIVA (Germany); SARA (France).
Educational	UoSAT (United Kingdom); SURFSAT-1, SQUIRT, CanSat, 3SAT (United States); BLUEsat (Australia), SUNSAT (South Africa).

SOURCE: Adapted from [Bolea-Alamanac \(2001\)](#).

### 2.2.5 Prospects for Small Satellites

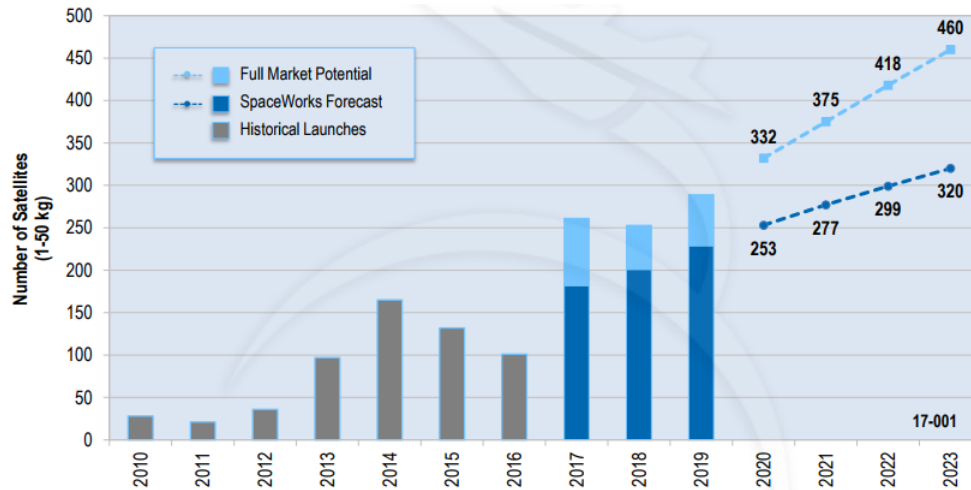
The proliferation of technology has been producing a world filled with scientific possibilities and challenges. The advancement of technology has potential to considerably increase the capacity of small satellites to accomplish significant missions on a low cost ([COUNCIL, 1994](#)). The decrease in cost and the better performance has begun an extensive succession of space missions. According to ([HEIDT et al., 2000](#)), the “smaller, cheaper, faster, better” philosophy has gained momentum in projects of satellites.

The substantial increase in the number of small satellites was mainly due to projects developed in universities for educational purposes. Space agencies, such as NASA and ESA also began to use small satellites as platform for new technologies, due to its low cost and short development time and especially due to the success in launches and in the operation of the first missions using this concept.

The 2017 market research for these satellites shows an important increase in the number of future projects for small satellites. Figure 2.2 presents the projection for the increase in the number of satellites up to 2023. Another significant advance is the increase in the number of small satellites for the commercial sector. Figure 2.3

shows a growth trend by sector in satellites up to 50kg.

Figure 2.2 - Small satellites Launch History and Forecast (1-50 kg).



SOURCE: SPACEWORKS ENTERPRISES (2017)

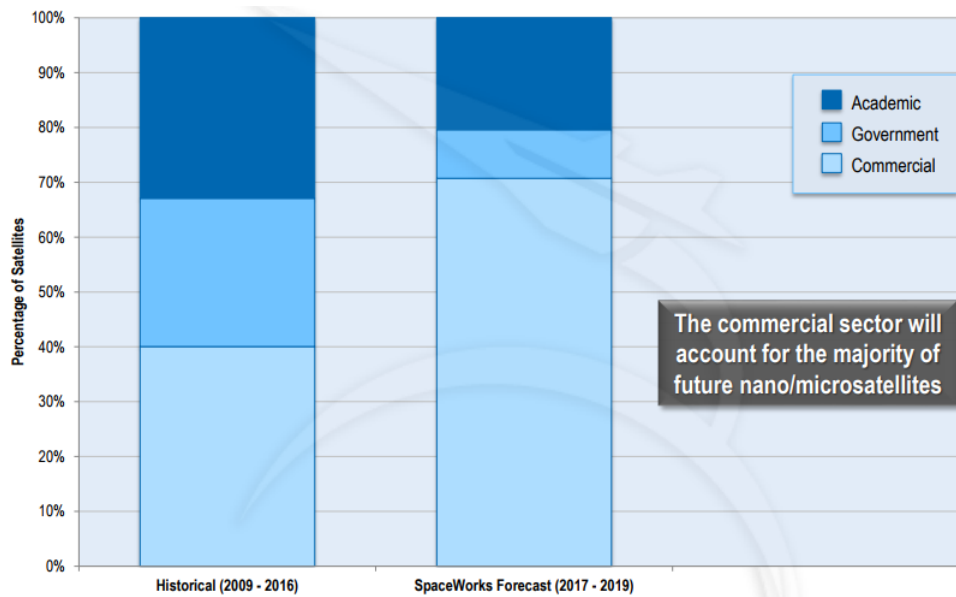
### 2.2.6 Small Satellites in Brazil

In Brazil, the first satellite to be designed, built and operated by INPE, the SCD-1, was launched in 1993, which weighed a total mass of 115Kg. Other projects for small satellites developed or still being developed are presented in Table 2.3.

## 2.3 Satellite Control

Orbiting satellites controlling is associated with these operating activities: attitude control, orbit control, service platform and payload control. Attitude control for satellites refers to the techniques employed to maintain, an acceptable range of values and the direction in which the satellite is pointed to in space. Attitude control in a satellite is made autonomously and independently by an electronic (usually based in microcomputers) on board the satellite. The service platform, payload and orbit control are made by operations on the ground. The ground segment refers to all ground systems to support preparation and the performance of the activities of the mission remote operation (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2000).

Figure 2.3 - Small satellites trends by sector (1–50 kg).



SOURCE: SPACEWORKS ENTERPRISES (2017)

Table 2.3 - Small satellites - Brazil

Satellite	Institution	Launching Year
SCD-1	INPE	1993
SCD-2	INPE	2007
SCD-2-A	INPE	2008
SCD-1	INPE	1993
NANOSATC-BR1	INPE/UFSM	2014
NANOSATC-BR1	INPE/UFSM	2016
AESP	INPE/ITA	2015
CONASAT	INPE/UFRN	2016
SERPENS	UnB	2015
Tancredo-I	Municipal School Tancredo Neves/INPE	2017
ITASat-1	ITA/INPE	2018

SOURCE: Author

The ground segment control contains all the equipment, services and programs needed to perform the control of the satellite during its life cycle in space and also the planning and the execution of the mission. In ground segment, the control

of the satellite is performed by ground stations and the Satellite Control Center (SCC).

The communication between the satellite and the ground segment is done when there is visibility of the satellite in its pass. The communication happens by sending packages of remote control and by the reception of telemetry packages from the platform and the payload. A telemetry package from the platform contains information on the status, temperature, electric tension levels, location, subsystems status, operations modes and current functioning parameters of the satellite. The telemetry package from the payload contains specific information of the space mission, such as imaging raw data and values read by testing sensors.

All the communication from the ground to the orbiting satellites is done by ground stations, which constitute the direct interface with the space section. A ground station is constituted mainly by UHF and VHF antennas, S band, microwave equipment, receivers, transmitters, digital equipment, communication equipment and computers. The main goal of the ground station is the establishment of communication between the ground control system and the satellites controlled by it, during the period when these satellites overfly their antennas' visibility range ([GALSKI, 2012](#); [ROZENFELD et al., 2002](#)).

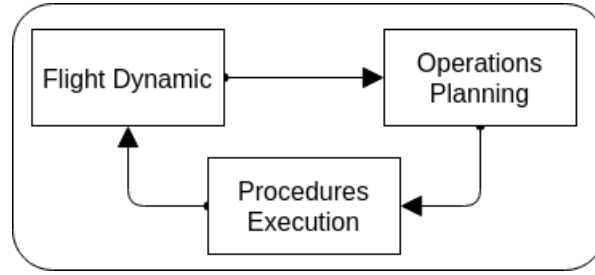
A ground station, through its antennas, must follow the satellite during its pass over the station. It must also receive, demodulate and record the telemetry data from the satellite (from the platform and the payload) and forward it to the SCC. It must radiate the remote control received from the SCC to the satellites in the scheduled time, measure speed and location (the distance between the satellite to the station and the azimuth and elevation angles from the satellite relative to the station) ([EUROPEAN COOPERATION FOR SPACE STANDARDIZATION \(ECSS\), 2000](#)), forward these measurements to the SCC, so they can be processed to determine the orbital control of the satellite.

### **2.3.1 Satellite Control Center**

According to [Ferreira \(2001\)](#), the SCC is the brain of all of the ground system. Its main function is to secure the normal working of the satellite since it is being placed in orbit until the end of its life cycle. Therefore, the SCC takes the vital role in the tracking and control of the satellites, having as main associated activities load control (platform and payload), attitude control and orbit control of the satellites.

The main satellite control operations are constituted by flight dynamics, operations planning and procedures execution. These operations are executed periodically and independently for each satellite. Figure 2.4 presents the relationship between these operations.

Figure 2.4 - Sequence of the Control Operations.



SOURCE: Tominaga (2010)

The flight dynamics is responsible for the definition and radiation of the orbital positioning and the attitude pointing information, using tracking and calibration measures. Operations' planning schedules the tasks to be done allowing the performing of the mission operations of each satellite. The procedures execution has the goal of performing the scheduled tasks in the operations planning (TOMINAGA, 2010). Control activities of satellites are scheduled as emergency activities. They are specific for each satellite and are changed according to the operation mode of the satellite.

The Satellites Control Center must have a minimum infrastructure to allow its functioning; having real-time software to track the equipment in the satellite; receiving data and radiating remote control. It must also have real-time software to track all the ground segment equipment and satellite simulators to diagnose possible problems and to validate the procedures.

Software systems for the SCC are complex and must execute various functions. Some of its main functions are: (i) to receive, to process and to store telemetry; (ii) to calculate maneuvers; (iii) to generate, to validate and to forward the remote control to the stations; (iv) to configure the operation of the payload; (v) to determine and to radiate the orbits and the attitude of the satellites; (vi) to receive and to process the measures of distance, speed and the angles; (vii) to receive and to track the

ground segment status; (viii) to store, to retrieve and to present historical data; (ix) to prepare and to perform maneuvers with the satellites; (x) to generate the passing projections for satellites over the stations; and (xi) to generate flight plans for the controlled satellites. The CCS also performs preparation stages to control and to support launches, involving adaptation of the ground infrastructure (ROZENFELD et al., 2002).

### 2.3.1.1 INPE's Satellites Control Center

The system of Satellite Tracking and Control Center (CRC), of the INPE located in São José dos Campos (SP) is constituted by the CCS; remote ground stations located in Cuiabá (MT) and Alcântara (MA). The CRC also have data and voice communication networks linking these places, which operates 24 hours per day, 365 days per year (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS, 2017). Through this network, satellite signals are received and sent to the CCS, where telemetry messages contained by the signal are decoded, seen in real-time and stored in digital format. On the opposite side, the remote control generated by the CCS is transmitted to the station which, in its turn, transmits it to the satellite. To control its activities, the CCS has the software *SATellite Control System* (SATCS), which was developed especially to control satellites developed by the INPE.

## 2.4 Software Architecture

The concept of software architecture is presented in the literature through various definitions. These definitions usually include words like *structure*, *connections*, *elements* and *components*. The following are some of these definitions.

Software architecture can be defined as a structure, or set of system structures, which includes software elements; externally visible properties of these elements; and the relationships between them (BRITO et al., 2007). In Bass et al. (2003), software architecture is defined as a high-level structure of the system in terms of architectural elements, abstracting details from its implementation. In Garlan (2000), the architecture is: the structure of components of a system; the relationships between these components, the principles and guidelines that govern the projects; and the evolution of the software. The ISO/IEC 42010 (INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2007), defines architecture as the fundamental organization of a system, represented by its components, relationships, and the principles that drive its design and evolution. On the website of the Software Engineering Institute (SEI), there is a list of definitions with definitions accepted by the software

community for the definition of the software architecture term<sup>1</sup>.

Software architecture becomes very important in the development process, contributing as a communication guide among project stakeholders, manifesting design decisions in advance, and facilitating the transfer of knowledge from the system. In the process of developing architecture, elements are organized to represent a system. When the architecture is defined for a given application, this architecture is termed as an *architectural instance*.

The development of a software architecture is the activity that determines the quality and maintainability of the software (WASSERMAN, 1996). During development, its components have the function of processing and storing the data for a certain functionality of the system (PERRY; WOLF, 1992). In addition to the components, another element of the architecture is the *connector*, which is responsible for the communication between the architectural components. A set of architectural components connected by architectural connectors define an *architectural configuration* (KRUGER; MATHEW, 2004).

#### 2.4.1 Architectural Style

In the development of a software architecture, some structural patterns are used, which determine how the elements will relate. The type of communication and the data inputs and outputs for each element in the architecture are also defined. Thus, this process of establishing the type of structures, communication and interfaces to be applied in software architecture, defines its architectural style. Architectural style can also constrain the kind of design elements and the formal relationships among them.

An architectural style defines a family of systems in terms of a pattern of structural organizations. It is an abstraction for a set of architectures that have a set of common architectural features. According to Zhu (2005), a architectural style is determined by the following types of features: a set of components; a set of connectors; a topological structure; and a set of semantic constraints. In Monroe et al. (1997), architectural styles have important properties, such as a vocabulary of high-level design elements; design rules for component compositions; and semantic interpretation. In order to present recurring solutions for the design of architectures, some styles were proposed, in the software engineering community. In the following, there is described some of the major architectural styles. The classification of the

---

<sup>1</sup><https://www.sei.cmu.edu/architecture/start/glossary/community.cfm>

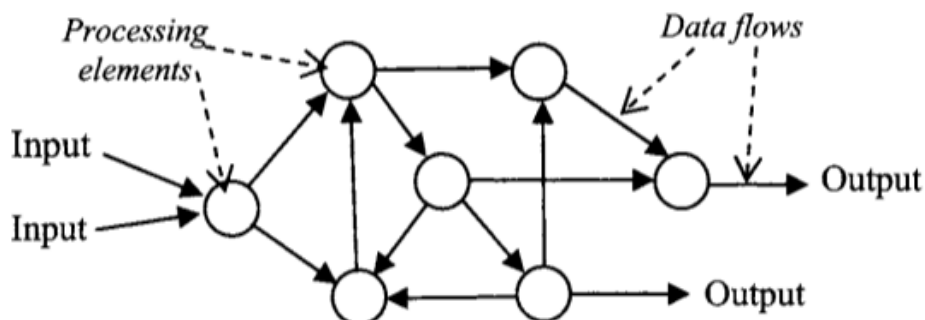


styles presented is based in (ZHU, 2005), a more complete set of architectural styles can be obtained in (GARLAN et al., 2010; SHAW; GARLAN, 1996).

#### 2.4.1.1 Data Flow Architectural Style

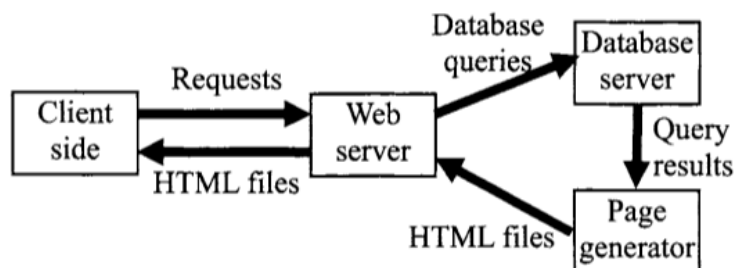
This style is widely used in various application domains where data processing plays a significant role. It is characterized by viewing the system as a series of transformations on successive pieces of input data. The structure of the design is dominated by orderly motion of data from component to component. A data flow system can be viewed as a directed graph, where nodes are processing elements and arcs are data flows between the elements (ZHU, 2005). Figure 2.5 illustrates the data flow style. Web-based applications can be regarded as in the data flow architectural style. For example, a typical e-business system depicted in the following Figure 2.6.

Figure 2.5 - Illustration of data flow style



SOURCE: Zhu (2005)

Figure 2.6 - The structure of a typical web-based application



SOURCE: Zhu (2005)

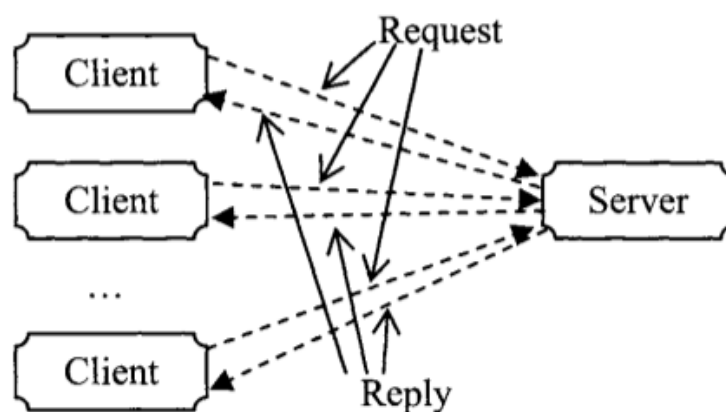
A way of understanding what can be different for systems in a style is to recognize its sub-styles. There are two main sub-types of data flow style, *sequential processing* and *pipe-and-filter*.

#### 2.4.1.2 Independent Component Architectural Style

This architectural style consists of a number of independent components that communicate asynchronously via messages. The message may be passed to named participants or passed among unnamed participants in case of using the publish/subscribe paradigm. The components send data to each other, but typically do not directly control each other. This style has attracted increasing interest recently for its strong support to software reuse and evolution due to the ease it provides for the integration of components into a system (ZHU, 2005).

The independent component style does not present a general pattern of system topological structure. A typical example of independent component architecture is the client-server architecture, which consists of a number of clients and one or more servers, as shown in Figure 2.7. This style has the following sub-types of style: *communicating processes*; *event-based implicit invocation*; and *multi-agent systems*. In addition to these sub-types, there is also C2 sub-style, which is designed to support the particular needs of applications that have a graphical user interface aspect. However, this sub-style clearly has the potential for supporting other types of applications (TAYLOR et al., 1996).

Figure 2.7 - Client-server structure



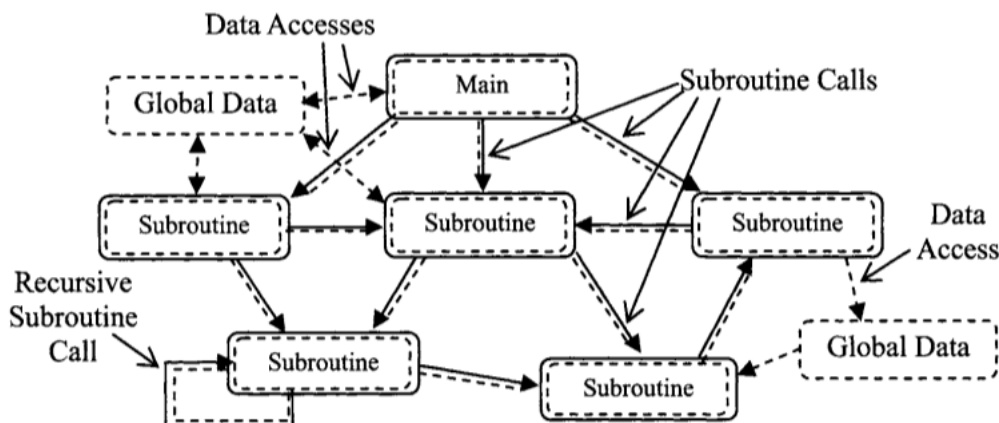
SOURCE: Zhu (2005)

### 2.4.1.3 Call and Return Architectural Style

In software development, it is necessary to partition the system into cohesive and weakly coupled subsystems (LARMAN, 2005). Subsystems are usually identified by the services they offer, defined through the operations of their interfaces. A software system in *call and return style* is essentially decomposed into smaller pieces to deal with complexity and to help achieve modifiability.

Call and return style are directly supported by almost all high-level programming languages with the procedure/function facility, also called subroutine. Figure 2.8 illustrates a structural pattern of a system in the *call and return style*. It can have any topological structure that links subroutines by subroutine calls.

Figure 2.8 - Structure pattern of call and return architecture

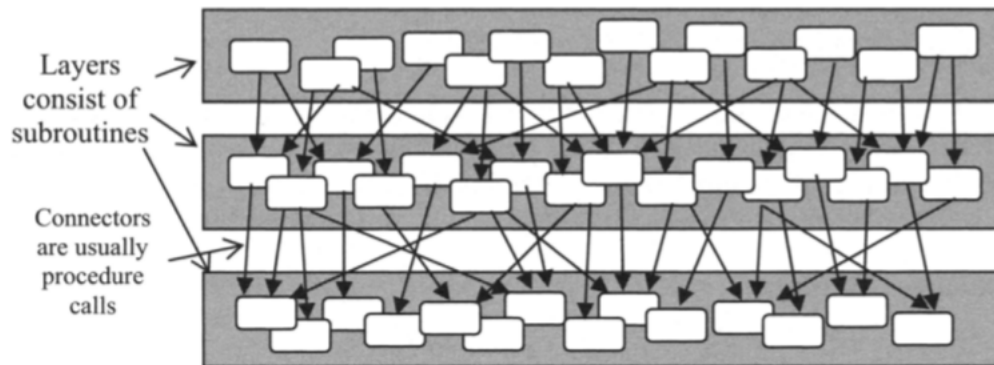


SOURCE: Zhu (2005)

There are two main sub-types of *call return style*: one is layered systems, the other is data abstraction including abstract data types and its advanced form object-oriented systems. The main characteristic of the layered sub-style is its way of organizing subroutines into a simple topological structure. Figure 2.9 shows a layered system. The subroutines are organized into a number of groups, which are called layers. Each layer contains a collection of subroutines that provide services at a certain abstract level to the layer or layers above it. Layers are widely used for building web systems, which consists of using three layers: (i) one responsible for the interaction between the system and the user; (ii) another responsible for implementing the business rules; and (iii) a third which handles the storage of data. Software architectures organized

according to this division are known as *three-tier architectures*.

Figure 2.9 - Structure pattern of layered systems



SOURCE: [Zhu \(2005\)](#)

#### 2.4.1.4 Data-Center Architectural Style

The data-center architectural style refers to systems in which the access and update of a widely accessed data-store. This style has two sub-types: repository and blackboard. The repository style is a system that allows several interfacing components to share the same data. The blackboard style sends notification to subscribers when data of interest changes and components are communicated exclusively through the repository.

#### 2.4.1.5 Virtual Machine Architectural Style

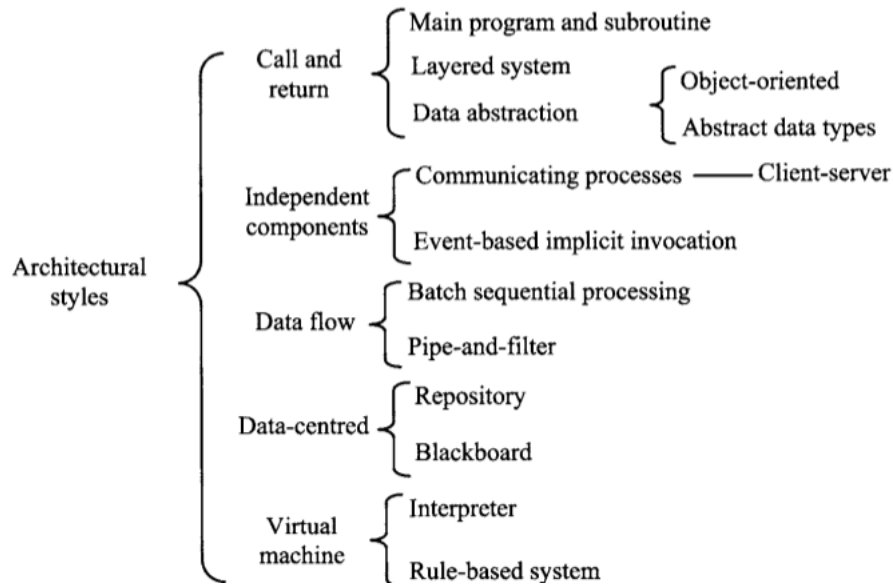
The virtual machine architectural style simulates a computer system, but with some functionality that is not native to the hardware and/or software on which it is implemented ([ZHU, 2005](#)). This style has two sub-types: interpreters and rule-based systems.

The interrelationships between styles and sub-styles are shown and summarized in Figure [2.10](#).

### 2.4.2 Architectural View

Structural aspects always receive greater prominence in the architecture definitions, but other aspects must be considered for a better understanding of the architecture.

Figure 2.10 - Summary of software architectural styles and sub-styles.



SOURCE: [Zhu \(2005\)](#)

In [Garlan et al. \(2010\)](#), for the complete understanding of the architecture of a software, it is necessary to consider not only the structural perspective, but also aspects such as physical distribution, communication process, and synchronization. Thus, the software architecture must be seen and described under different views that must define its components, relationships, interactions, properties, characteristics, and constraints.

A architectural view is the representation of a coherent set of architectural elements, duly written and read by system stakeholders. It consists of a representation of a set of elements and the relations among them. One of the models proposed in the literature to organize the different architectural view is the model proposed by ([KRUCHTEN, 1995](#)), known as the “4+1 View Model of Architecture”. In this model the software architecture is defined in the five competing views, which are detailed in the sequence:

- a) Logical View: a description of the data within the system and the decomposition of the system into logical abstractions. The behavior is distributed among these abstractions;
- b) Process View: a description of the organization of the data elements, in

terms of process and threads executing that will run on the hardware;

- c) Development View: focuses on the organization of software in relation to the development environment. In this view, the elements are libraries, sub-systems, files, class, and others;
- d) Physical View: also known as the deployment view, this view shows the physical organization of the system. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components;
- e) Scenarios View: used to illustrate the behavior of the system in its architecture through the main scenarios of use cases.

Another proposal that presents the architecture through views is the model defined by (HOFMEISTER et al., 1999). In this proposal, the classification of the description of a software architecture is presented in four views: conceptual, module, execution, and code. Because of these four views, the model is called “The 4 Views Architectural Model”.

In Herzum and Sims (2000), the following views are characterized as the most relevant visions for definition of software architecture:

- a) Technical Architecture: refers to the components of the execution environment, tools and other technical facilities required to develop and run a component-based system;
- b) Application Architecture: refers to the set of architectural decisions, standards and guides required to build a system based on components;
- c) Project Management Architecture: presents concepts, guides, principles and management tools needed to build a large system;
- d) Functional Architecture: where the specification and implementation of the system is performed.

In Clements (2003), three categories of views are presented to represent the software architecture: *Module View*, *Component-and-Connector View*, and *Allocation View*. These views are detailed in the sequence:

- a) Module View: it considers the elements as modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned responsibilities or functionalities;
- b) Component-and-connector View: elements are run-time components (main computing units) and connectors (which represent communication paths among components);
- c) Allocation View: this view presents the relationship between the elements of the software and the elements of the external environments where the system is executed.

### 2.4.3 Architectural Representation

Architectural representation plays an important role in communication, quality assessment, evaluation, and evolution in software architecture. The architecture of a system is usually expressed through a set of diagrams representing different architectural views of the system.

However, considering the importance of the software architecture, it is essential to properly represent software architecture during the life-cycle of software systems.

Software architecture can be represented in different forms. An Architectural Description Language (ADL) can define different forms of representation of a software architecture ([INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2007](#)). As examples of ADLs, we can mention the languages: Architecture Analysis & Design Language(AADL)<sup>2</sup>, Systems Modelling Language (SYSML)<sup>3</sup>, Unified Modeling Language (UML)<sup>4</sup>, ArchiMate<sup>5</sup>.

Among the existing architectural description languages, it is important to highlight the UML. The UML has been the basis for modeling many software systems, it has a wide variety of diagrams and also various tools that support its adoption at different stages of the software life cycle. From its version 2.0, in addition to system modeling, UML presented technical characteristics and capabilities for architectural software description ([IVERS et al., 2004](#)). In its version 2.5, the UML provides a set of diagrams divided into three categories (Structure, Behavior, Interaction), as shown in Figure 2.11.

---

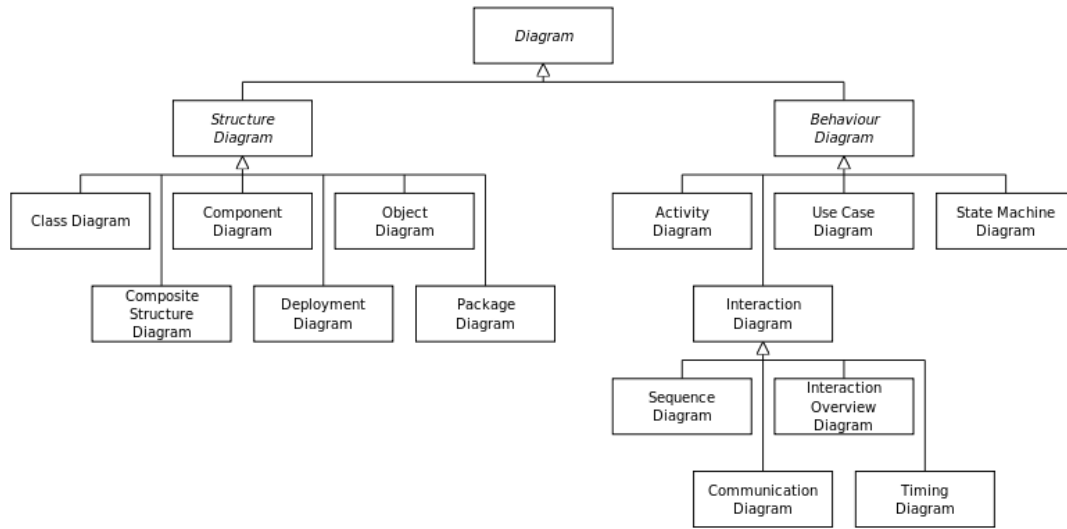
<sup>2</sup><http://www.aadl.info>

<sup>3</sup><http://www.omg.sysml.org>

<sup>4</sup><http://www.uml.org/>

<sup>5</sup><http://www.opengroup.org/subjectareas/enterprise/archimate-overview>

Figure 2.11 - Diagram UML Version 2.5.



SOURCE: UML... (2015)

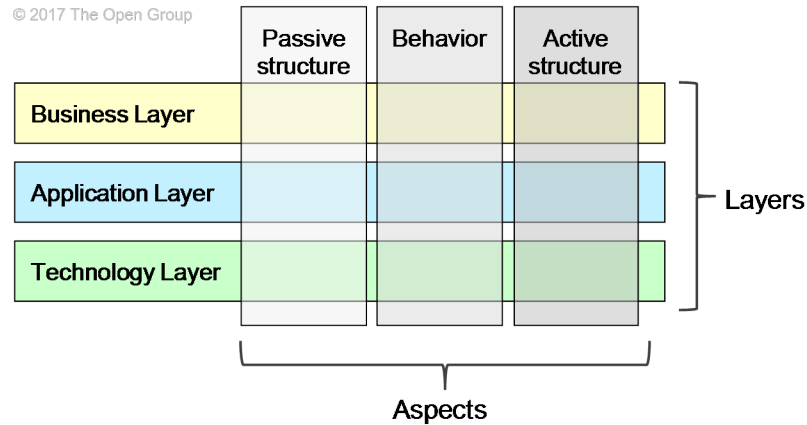
The ArchiMate is an enterprise architecture modeling language based on the IEEE 1471 standard (INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2000). It is maintained by the Open Group<sup>6</sup>. ArchiMate supports the description of the construction and operation of business processes, organizational structures, information flows, IT systems, and technical infrastructure. The latest version of the language is the ArchiMate 3.0 Specification.

ArchiMate defines a structure of generic elements and their relationships. It consists of three layers(Business, Application, and Technology) and three aspects(Active Structure, Behavior, and Passive Structure). Figure 2.12 shows the relationship between layers and aspects. The Business layer is typically used to model the structure and interaction between the business strategy, organization, functions, business processes, and information needs. The Application layer is typically used to model the information systems architectures of the enterprise, including the structure and interaction of the applications. The Technology layer is typically used to model the technology architecture of the enterprise, which consists of the structure and interaction of the platform services, and logical and physical technology components.

<sup>6</sup><http://www.opengroup.org/>



Figure 2.12 - Archimate - Layers and Aspects



SOURCE: OpenGroup (2017)

## 2.5 Reference Architecture

The concept of software architecture as a particular discipline has begun to emerge since the 1990s. It had strong growth with contributions from the industry and from the academia (SHAW; GARLAN, 1996). From then on, there were also highlighted the growth of interest on the topic of reference architecture (KRUCHTEN et al., 2006).

Despite the growing interest and spread of reference architecture, there is still no consensus on its exact definition of the topic (ANGELOV et al., 2009; NAKAGAWA et al., 2013). The following are the main definitions about this topic.

- The Reference architecture refers to an architecture that covers the knowledge about how to design concrete architectures of the systems for a given application domain. It refers to a special type of software architecture that captures the essence of the architectures of a set of systems in a given domain (OUSSALAH, 2014).
- The reference architecture is, in essence, a standard or a set of predefined architecture patterns. It is designed and approved to be partially or fully instantiated in certain technical or business contexts (KRUCHTEN, 2000).
- The reference architecture is a generic architecture for a class of information systems, which is used as the basis for the development of concrete architectures (ANGELOV et al., 2009).

- d) The reference architecture consists of a repository of domain knowledge that promotes architectural reuse and supports the development of systems (BASS et al., 2003).

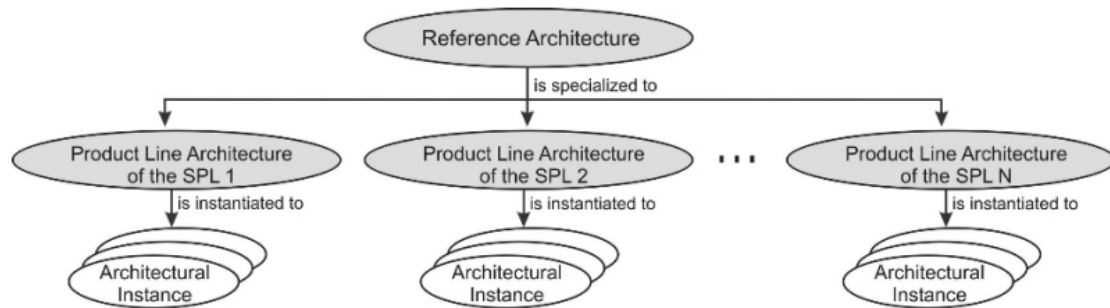
The 'reference architecture' term is generally considered synonymous with the term 'product line architecture' when assigned in the context of domain engineering, especially in the design phase during the creation of the architecture artifact. However, despite the similarities of the terms, there are several differences between the reference architecture and the product line architecture (NAKAGAWA et al., 2011):

- a) Product line architectures are more specialized, sometimes focusing on a specific subset of one domain system and providing standardized solutions for a smaller family of systems;
- b) Reference architectures are used to standardize the external structure of the systems (interfaces and protocols) and consider the subsystems involved as black boxes. However, in product line architecture the focus is to build systems from a common set of artifacts. The internal structure is standardized, aiming at the reuse of existing components that are considered white boxes. Thus, reference architectures focus on external standardization, while product line architectures address internal standardization.
- c) Product line architectures work with *variabilities* (information on how each product differentiates itself into a software product line).

Typically, reference architectures have a higher level of abstraction if compared to the product line architecture, so they can also be the basis for product line architectures. In Nakagawa et al. (2013), a process is presented to systematize the use of the reference architecture in the context of software product lines, this relationship is shown in Figure 2.13.

Reference architecture is the abstraction of architectural knowledge for a particular domain (OUSSALAH, 2014; MARTÍNEZ-FERNÁNDEZ, 2013), providing major guidelines for the specification of software architectures (NAKAGAWA et al., 2011) that constitutes the main artifact to the success of software systems (KRUCHTEN et al., 2006; SHAW; CLEMENTS, 2006). Among the several goals of the software reference architecture is to serve as a guideline for the development, standardization and evolution of several software architectures of a specific domain (NAKAGAWA et al.,

Figure 2.13 - Relationship between reference architecture and product line architecture.



SOURCE: Nakagawa et al. (2014)

2011). This is possible because the software reference architecture is abstract enough to allow its use in different contexts (ANGELOV et al., 2012).

Since Reference architecture is an important area of research in software architecture, several studies have focused on this special architecture type. Diverse domains, both in academy and industry, understood the advantages of such architecture and already possess their own. Examples of such domains are Automotive (MARTÍNEZ-FERNÁNDEZ et al., 2015), Smart City (CLEMENT et al., 2017), Ambient Assisted Living (FERRO et al., 2015), Big Data (KLEIN et al., 2016), On-board Space Applications (PANUNZIO, 2011), Software Engineering (NAKAGAWA et al., 2011), Embedded Systems (GUESSI et al., 2015), Internet of Things (WEYRICH; EBERT, 2016), Healthcare Supportive Home Systems (RODRÍGUEZ et al., 2015), Mobile Learning Environments (FILHO; BARBOSA, 2015), Recommender Systems (SHETH et al., 2010), and many others.

There are several types of reference architecture. A classification is presented in Angelov et al. (2012), which defines that the reference architecture can be classified through multidimensional space, based on the dimensions of context, goals, and design. The following classification of the reference architecture defines them as:

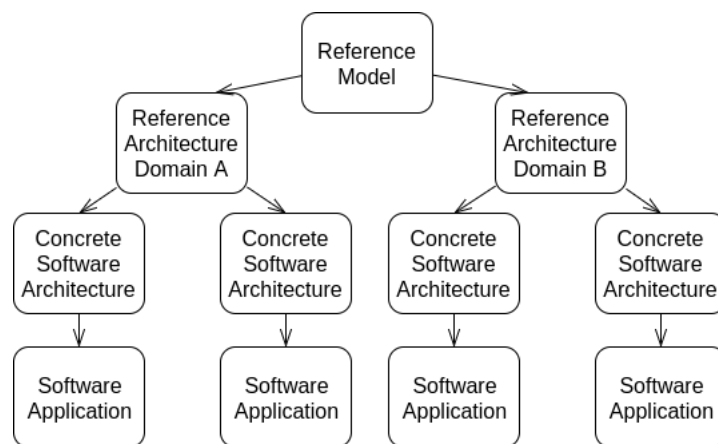
- a) Type 1: These are classical, standardized architectures, designed to be implemented in multiple organizations, and have a representative set of users and software.
- b) Type 2: These are classical, standardized architectures, designed to be implemented in an organization. Groups of software users provide require-

ments that designers of the software design groups.

- c) Type 3: These are classic facilitation architectures, designed by an independent organization to be implemented in multiple organizations.
- d) Type 4: These are classic facilitation architectures, designed to be implemented in a single organization. Their representation may be semi-formal or informal.
- e) Type 5: These are preliminary architectures, designed to be deployed in multiple organizations to facilitate future architecture projects.

Building a RA involves several steps and activities and should address several issues such as business rules, architectural styles and patterns, architectural representation, practices of software development, domain constraints, legislation and standards (NAKAGAWA et al., 2011). A Reference model intends to improve the understanding about their elements and relationships, supporting the establishment, use, and evolution of such architectures. Figure 2.14 illustrates the relationship between the reference model, the reference architecture, the concrete architectures and software application.

Figure 2.14 - Relationship between reference model, reference architecture, concrete software architectures and software application.

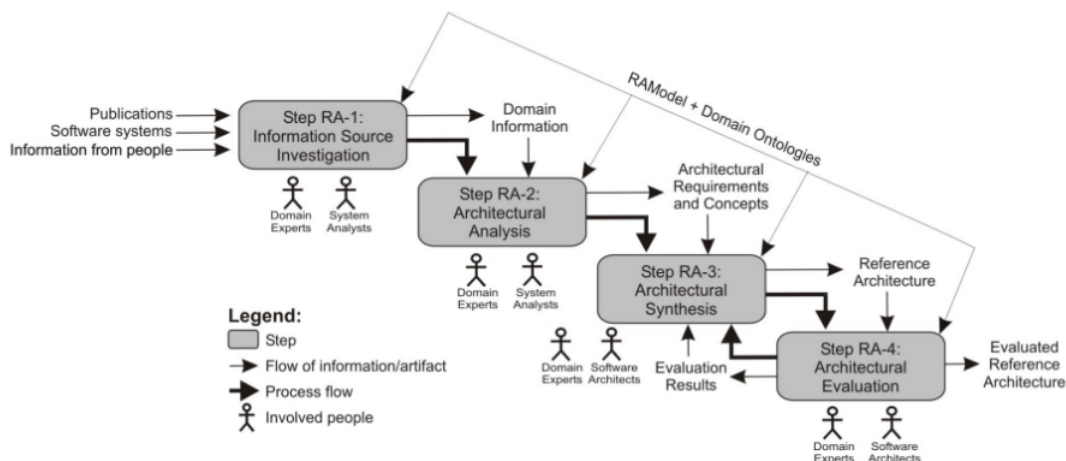


SOURCE: Author

### 2.5.1 ProSA-RA: A Process to Build Reference Architectures

Diverse works have focused on the building of reference architectures (NAKAGAWA et al., 2014; ANGELOV et al., 2012; BAYER et al., 2004; CLOUTIER et al., 2010; DOBRICA; NIEMELÄ, 2008) and their validation (NAKAGAWA et al., 2012; ANGELOV et al., 2008; GALLAGHER, 2000; GRAAF et al., 2005; SANTOS et al., 2013). We adopted the systematic process ProSA-RA Nakagawa et al. (2014) to establish the SCS-RA, considering that it prioritizes the design, representation, and evaluation of reference architectures and has been already applied in the establishment of many reference architectures (NAKAGAWA et al., 2014). ProSA-RA, illustrated in Figure 2.15, is composed of four steps (NAKAGAWA et al., 2014): Information Source Investigation, Architectural Analysis, Architectural Synthesis, and Architectural Evaluation. These steps are detailed in the sequence.

Figure 2.15 - Outline structure of ProSA-RA.



SOURCE: Nakagawa et al. (2014)

#### 2.5.1.1 Step RA-1: Information Source Investigation

This step aims to search various sources of information from the domain. There are a number of ways that you can use to gather relevant information, books, articles, existing software architectures, existing systems, knowledgeable people (e.g., users, researchers, experts) and domain ontologies. These sources should provide information about the process, activities and tasks to be automated. Revisions could be made to systematize the process of searching, selecting and analyzing sources. Do-

main experts and system analysts actively participate in this step. As output from this step, a group of information sources is presented.

#### **2.5.1.2 Step RA-2: Architectural Analysis**

Based on the information sources selected in the previous step, this step performs the activities of identifying system requirements, establishing the requirements of the reference architecture and identifying the domain concepts. System analysts and specialists are directly involved in these activities. As outputs of this step, a set of reference architecture requirements and a set of domain concepts are generated.

#### **2.5.1.3 Step RA-3: Architectural Synthesis**

In this step, the design of the reference architecture is carried out, based on the architectural requirements and concepts identified in the previous step. An architectural description of the reference architecture and its representation through architectural views is carried out. During this step, the following viewpoints are proposed to describe the architecture: crosscutting, runtime, deployment and source code. Software architects, supported by domain experts participate in this step. In the end, this step results in a description of the architecture composed of a set of views of the reference architecture.

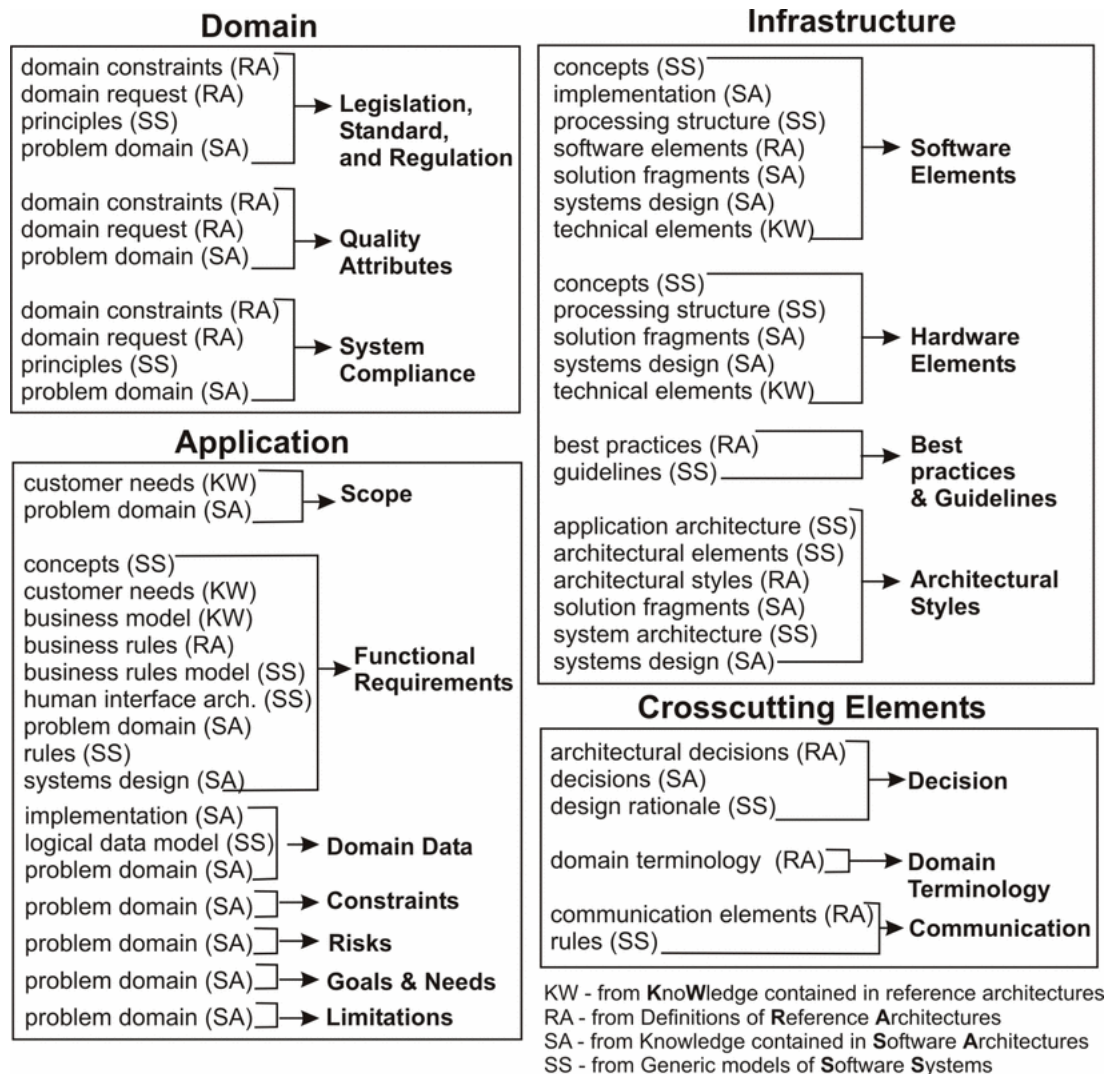
#### **2.5.1.4 Step RA-4: Architectural Evaluation**

In this final step, the objective is to verify the quality of the architecture through methods of evaluation of architectures, such as Software Architecture Analysis Method (SAAM) (GRAAF et al., 2005) and Architecture Tradeoff Analysis Method (ATAM) (GALLAGHER, 2000). The evaluation can also be performed by inspection using checklist through Framework for Evaluation of Reference Architectures (FERA) described in (SANTOS et al., 2013). Another way is using the Reference Architecture Model (RAModel), which is a reference model specific to reference architectures, providing a complete list of elements needed for constituting the reference architecture. These elements are classified in four groups (NAKAGAWA et al., 2012): Domain, Application, Infrastructure and Crosscutting. Figure 2.16 shows these groups and their elements.

### **2.6 Software Components**

A software component is the software element which provides the functional behavior of applications. Software components have a set of key features, such as:

Figure 2.16 - Groups and Elements of RAModel.



SOURCE: Nakagawa et al. (2012)

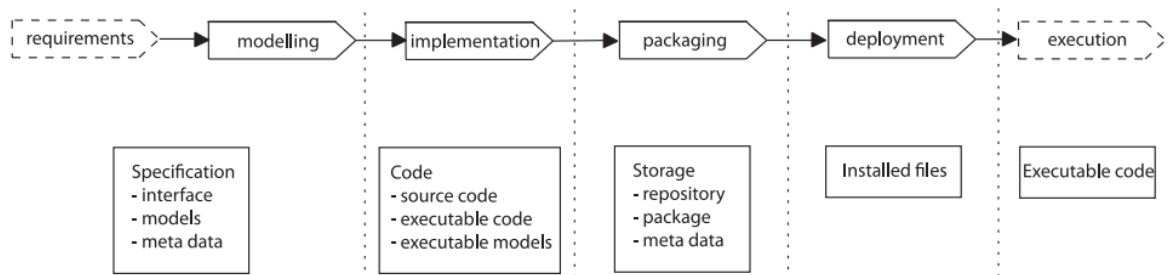
performing specific and well-defined functions (BROWN; SHORT, 1997; KOZACZYNSKI, 1999; SAMETINGER, 1997; YACOU B et al., 1999); providing a set of interfaces (SAMETINGER, 1997); and provides adequate documentation to contribute to the search, retrieval, adaptation and integration (HEINEMAN; COUNCILL, 2001; SAMETINGER, 1997; MCCLURE, 1997). A detailed study of the characterization of a software component is presented in (BROY et al., 1998).

In the building of a new software component, its development process is in many aspects similar to system development (CRNKOVIC et al., 2006). A study comparing these processes is presented in (CRNKOVIC et al., 2006). During the process of devel-



opment of a component, its requirements must be captured, analyzed and defined and the component must be designed, implemented, verified and validated. A life-cycle for the development of a software component (with these steps: requirements, design, implementation, deployment and execution) is presented in (CRNKOVIC et al., 2011). Figure 2.17 shows these successive stages of the lifecycle of a component. The lower part is presented the ways in which components may be represented in that particular stage of the lifecycle. Software components have evolved since they were

Figure 2.17 - Component lifecycle



SOURCE: Crnkovic et al. (2011)

mentioned in the NATO Software Engineering Conferences in 1968. This evolution occurs mainly due to technological advances in the Computer Science and Software Engineering areas. Software components changed from a scientific research field to a widely used technique. Components speed up the adoption of new technologies and improve reusability. The evolution in the development of software components occurs from the definition of the component models. A model component contributes in several aspects to the construction of a software component.

### 2.6.1 Components Model

A component model is defined by the characteristics of its implementation, interoperability, customization, composition, infrastructure, evolution and deployment. Currently, there is a variety of component models available; each model provides different approaches to building applications.

Component models are intended for general-purpose and specific domains. Several domains already have their software component models and these models meet their specified domain characteristics. Table 2.4 shows a list of component models and



their purpose.

Table 2.4 - Component Models and Purpose

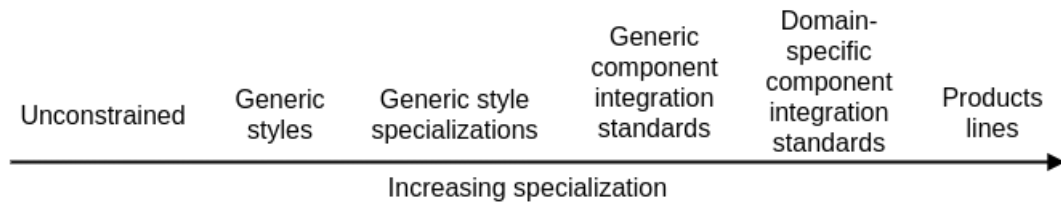
Component Model	Purpose
EJB ( <a href="#">ENTERPRISE...</a> , 2017), OSGi ( <a href="#">OSGI™...</a> , 2017) COM ( <a href="#">COM...</a> , 2017), .NET ( <a href="#">MICROSOFT</a> , 2017), CORBRA ( <a href="#">CORBA...</a> , 2016) ICEMI ( <a href="#">FURMENTO et al.</a> , 2001)	Software Application
Pebble ( <a href="#">MAGOUTIS et al.</a> , 2000), Koala ( <a href="#">OMMERING et al.</a> , 2000), PECOS ( <a href="#">GENSSLER et al.</a> , 2002)	Embedded Systems
THINK ( <a href="#">FASSINO et al.</a> , 2002), OSKit ( <a href="#">FORD et al.</a> , 1997), MMLite ( <a href="#">HELANDER; FORIN</a> , 1998)	Operating Systems
VERA ( <a href="#">KARLIN; PETERSON</a> , 2001), MicroACE ( <a href="#">JOHNSON; KUNZE</a> , 2003), NetBind ( <a href="#">CAMPBELL et al.</a> , 2002)	Programmable Networking Environments
LegORB ( <a href="#">ROMAN et al.</a> , 2000), K-components ( <a href="#">DOWLING; CAHILL</a> , 2004; <a href="#">GRIFFITH; KAISER</a> , 2005)	Middleware Platforms
ASSERT Project - Component Model ( <a href="#">PANUNZIO; VARDANEGA</a> , 2010)	On-board Software Space Domain
SaveCCM ( <a href="#">HANSSON et al.</a> , 2004)	Vehicular Systems

SOURCE: Author

In the study conducted by [Crnkovic et al. \(2011\)](#), it is possible to see that approximately 50% of the component models are domain-specific models. Domain-specific component models offer advantages of standardization, integration, reuse and interoperability. In addition to this, they may be considered an architectural trend. In [Garlan et al. \(2009\)](#), the specialized architecture is one way to help prevent architectural mismatch. Figure 2.18 illustrates the architectural specialization space. At the far left are completely unconstrained architectures; to the right, the architectures must fit in a narrower design context.

For a better understanding of the characteristics of a software component, we present next a detailed description of some major component models.

Figure 2.18 - Architectural Specialization



SOURCE: [Garlan et al. \(2009\)](#)

### 2.6.1.1 Enterprise Java Beans (EJB)

EJB is a component model for building distributed applications, which adopts the Java programming language. Access to its interfaces is accomplished by invoking the remote method and by passing messages.

EJB model has three types of components: the Session bean, the Entity bean, and the Message-driven bean. The Session bean implements business logic; the Entity bean represents the domain entities and the Message-driven bean is responsible for handling asynchronous messages from clients.

EJBs run in the EJB container. Containers provide services of transactions, security, they maintain life cycle for the EJBs and delegate calls to them. Containers run in the Java EE servers. Compatible examples of these servers are Glassfish<sup>7</sup>, JBoss Enterprise Application Platform<sup>8</sup>, Wildfly<sup>9</sup>, Apache TomEE<sup>10</sup> and others.

### 2.6.1.2 CORBA Component Model (CCM)

The CCM is a model of the OMG's<sup>11</sup> component specification. The CCM specification defines an abstract model; a programming model; a packaging model; a deployment model; an execution model; and a 'metamodel' ([CRNKOVIC et al., 2011](#)). It also provides a Component Implementation Definition Language (CIDL); the semantics of the CORBA Components Model (CCM); and a Component Implementation Framework (CIF).

<sup>7</sup><https://javaee.github.io/glassfish>

<sup>8</sup><https://www.redhat.com/en/technologies/jboss-middleware/application-platform>

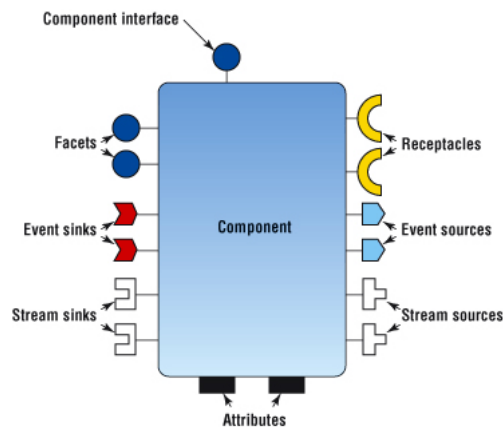
<sup>9</sup><http://wildfly.org>

<sup>10</sup><http://tomEE.apache.org>

<sup>11</sup><http://www.omg.org/spec/CCM/>

The CCM component has a single *equivalent interface* to clients; a set of *attributes* (which are typed parameters exposed through *getter* and *setter* operations); It also provides a set of *facets* (which are named ports providing specific interfaces); a *receptacle* port (which are named connection points that describe the component's ability which is possible to invoke operations of other components); a *event sources* (which are named connection points that send events according to publish-subscribe paradigm); a *event sinks* (which are named connection points into which events of a specified type may be pushed. It consumes events *event sources*); a *stream source* port (which produces streams of a specific type of data), and a *stream sink* port (which receives such data). Figure 2.19 depicts a CORBA component's features.

Figure 2.19 - CORBA Component Model



SOURCE: Crnkovic et al. (2011)

CCM is non-proprietary language and platform independent. It allows designing and implementing a distributed application independent of specific programming languages, operating systems or vendor-specific communication infrastructures (BORN et al., 2003).

### 2.6.1.3 Open Services Gateway Initiative (OSGi)

OSGi<sup>12</sup> is a consortium of numerous industry partners who work together to define a Java-based interface specification (OSGi™..., 2017). OSGi defines a standardized, component-oriented, computing environment for networked services. It has implementations from organizations, like the Eclipse Foundation, with their Equinox

<sup>12</sup><https://www.osgi.org>

framework (MCAFFER et al., 2010) and the Apache foundation, with the framework Felix (GÉDÉON, 2010).

The OSGi framework separates the components for deployment and run-time into bundles and services respectively. A bundle is comprised of java classes and other resources, which together provide functions to end users.

OSGi is specific to Java. It is not platform-specific, requiring runtime container to manage the component registration. The components are independent and can be added, removed, or modified at runtime.

## 2.7 Component-Based Development

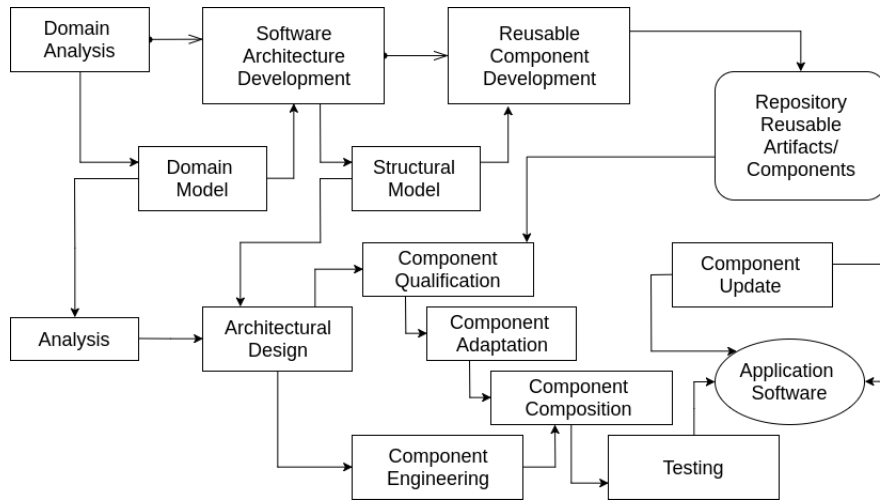
The Component Based Development (CBD) is a software development technique that builds on the rapid building of systems from existing components (BACHMANN et al., 2000). A system is developed not as a monolithic entity, but from the composition of parts that have already been constructed separately. The concept of reusing parts of existing systems arose primarily to amortize development costs, to reduce complexity in software development and to increase the quality of systems produced (SZYPERSKI, 2011).

An important feature of CBD is the separation between the process of the development of the general system and the process of the development of the individual components. The CBD, like any other software development process, involves developmental phases with activities resulting in software artifacts, the difference being that the system is constituted with the base unit of components. In Brown and Short (1997), McClure (1997), Pour (1998), the predominant phases in the process models of CBD are: analysis of requirements, acquisition, component understanding, adaptation, composition and certification of the component. These phases are also present in the process model proposed by (PRESSMAN, 2001), which is shown in Figure 2.20

The main activities of component-based development are the creation of components to be reused and the development of software based on existing components (KRUEGER, 1992). In Kotonya et al. (2003), these activities are defined as *component engineering* and *application engineering*.

Component engineering aims to develop reusable components for the construction of systems. The main steps are: domain analysis, component building, maintenance and publication, and certification of software components relevant to a domain.

Figure 2.20 - Process Model of CBD



SOURCE: Pressman (2001)

Application engineering aims to apply the software components in the development process. Its main responsibilities are to qualify, adapt and integrate components for use in new systems. In Brown and Short (1997), there are five steps in this process: selection, qualification, adaptation, composition and updating.

### 2.7.1 Component-Based Development Process

With CBD gaining acceptance and other approaches, such as software reuse and software product line, several component-based development process have emerged. Among these processes we can highlight: The Catalysis approach (D'SOUZA; WILLS, 1999), FORM: A Feature-oriented Reuse Method with Domain-specific Reference Architectures (KANG et al., 1998), RiDE: The RiSE Process for Domain Engineering (ALMEIDA, 2007), and CoPAM: Component-Oriented Platform Architecting Method (AMERICA et al., 2000). Next, we present a brief description of UML Components and Kobra process.

UML Components (an extension of the UML), represents a process of software specification based on components (CHEESMAN; DANIELS, 2001). The process uses a simple way to extend UML based on stereotypes. It provides a solution without specifying the platform, so that the method can be used on many platforms and with many different technologies.

In UML Components, the system architecture is structured into four distinct layers:

user interface, user communication, system services, and business services. In addition, the process is compatible with UML for modeling all phases of development, including activities such as requirements definition, identification and description of component interfaces, modeling, specification implementation and composition.

The KobrA approach ([ATKINSON et al., 2000](#)), which has been developed at the Fraunhofer Institute for Experimental Software Engineering (IESE), aims at providing a systematic approach to the development of high-quality, component-based application frameworks. KobrA has two main activities: initially, framework engineering creates and maintains a generic framework.

In KobrA, a framework is the static representation of a set of *Komponents* (KobrA component). Each Komponent is described at three levels of abstraction: specification, realization and implementation. For each level of abstraction there is a set of artifacts recommended for its specification as well as a set of activities associated with its production. Most of these specifications use UML diagrams and textual description. The complete list is presented in ([ATKINSON et al., 2002](#)).

### 3 RELATED WORK

This chapter presents the related work and introduces some important initiatives which encourage the creation of reference architecture for space systems. It explores some software application for Satellites Control Systems and their architectures. The European Ground Systems – Common Core (EGS-CC), Control Channel Toolkit (CCT), Hifly system, Satellite Control and Operation System 2000 (SCOS-2000), Goddard Mission Services Evolution Center (GMSEC), Global Educational Network for Satellite Operations (Genso), and Satellite Network (SatNet) are part of this chapter.

#### 3.1 Reference Architecture for Space Systems

CCSDS Recommended Practice entitled “Reference Architecture for Space Data Systems (RASDS)” ([THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008](#)) provides guidelines for the description of space data system architectures and high-level designs that take into account the issues of operations in the space environment. RASDS provides the structure for a full semi-formal representation of space data systems and also introduces a set of conventions for representing space data systems from several viewpoints<sup>1</sup>. The development of a semi-formal representation using SysML<sup>2</sup> is the subject of future proposed works ([THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008](#)). RASDS provides five specific and complementary viewpoints on the system and its environment ([THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008](#)): enterprise viewpoint, functional viewpoint, connectivity viewpoint, communications viewpoint, and information viewpoint.

RASDS is intended to be soon used by CCSDS to establish an overall methodology for defining and developing domain-specific architectures, as well as for defining a common language, a taxonomy, and representations. ESA is applying RASDS to a European technology harmonization to Ground Software Systems ([SHAMES; YAMADA, 2003; DURO et al., 2005; REID, 2012](#)). Another related work is shown in [Shames and Yamada \(2003\)](#) that presents how RASDS can be used to reduce the cost of development of space data systems. In addition, an architectural framework

---

<sup>1</sup>Software architectures are often structured in views and viewpoints. A view is often materialized in a model, and corresponds to one of the facets of a software. For example, a logical view (usually represented in class diagrams UML), or a physical view, often represented as a deployment diagram. In turn, a viewpoint corresponds to a collection of patterns, templates, and conventions for constructing one type of view ([KRUCHTEN, 1995; ROZANSKI; WOODS, 2005](#)).

<sup>2</sup><http://www.omg.sysml.org>

applied to RASDS is introduced in (DURO et al., 2005).

Another initiative of CCSDS supporting the practice of RA is shown in “Reference Architecture for Space Information Management (RASIM)” (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2013). RASIM is an informational report that presents a RA for space information management (information architecture). The information architecture covers problem areas associated with space data systems (such as organizational, functional, operational, and cross-support issues). RASIM is intended to provide an overview and background for those interested in understanding and developing information architectural elements for building space data systems. RASIM considers the lack of a real description of complex space data system architectural topologies and best practices of using software components.

RASDS and RASIM are high-level architectures designed to facilitate and encourage the creation of RA for space systems that will be needed in the future. In the context of space systems and RA, ESA has recently worked on the definition, realization, and adoption of a RA for the development of on-board software for satellites (PANUNZIO; VARDANEGA, 2013).

In short, we observe that RA is a real and important element for space systems. In the same perspective, the establishment of a RA for SCS is equally important.

### **3.2 Software Application for Satellites Control Systems**

Software applications have increasingly assumed an important role in the satellites control activities and have contributed in an effective way to meet the activities of this domain.

#### **3.2.1 The European Ground Systems – Common Core (EGS-CC)**

The European Ground Systems – Common Core (EGS-CC)<sup>3</sup>, is an ESA initiative with European agencies and industry to develop a common software infrastructure to support space systems monitoring and control for all mission types (EUROPEAN SPACE AGENCY (ESA), 2017a). The initiative is considered to be a strategic move which would bring an array of resources and benefits to all parties involved.

The EGS-CC is a platform on which a monitoring and control systems can be built and which provides core monitoring and control features as well as a set of components that allow adapting the core monitoring and control features to the operation

---

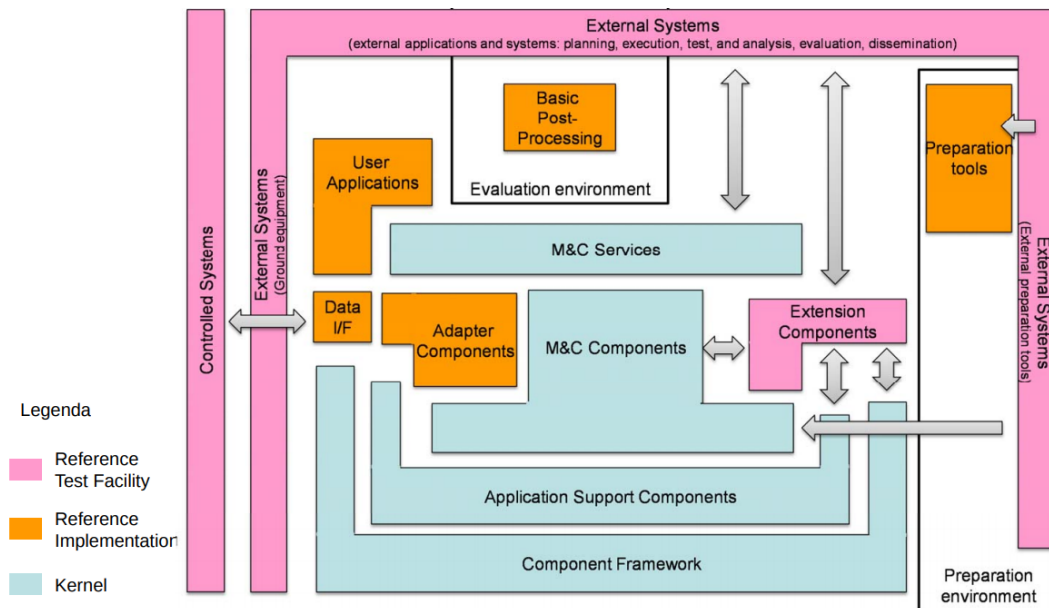
<sup>3</sup><http://www.egscc.esa.int>



environment (GOETZELMANN et al., 2014). It comprises of the software source code and binaries for a given EGS-CC based system. The EGS-CC defined design goals including: Open, component based, service oriented architecture; Support of all mission types; Generic and extensible functionality; Extensibility via binary interfaces; Inter-operability through standardised interfaces; High performance and scalability, and many others.

The EGS-CC architecture is logically structured in separate layers, isolating different responsibilities and defining clear interfaces between them. EGS-CC is divided into three architectural layers, *the Kernel*, *the Reference Implementations*, and *the Reference Test Facilities*. Figure 3.1 shows the architectural layers and high-level decomposition of the EGS-CC. The EGS-CC Kernel contains core monitoring and

Figure 3.1 - High-level Decomposition of the EGS-CC.

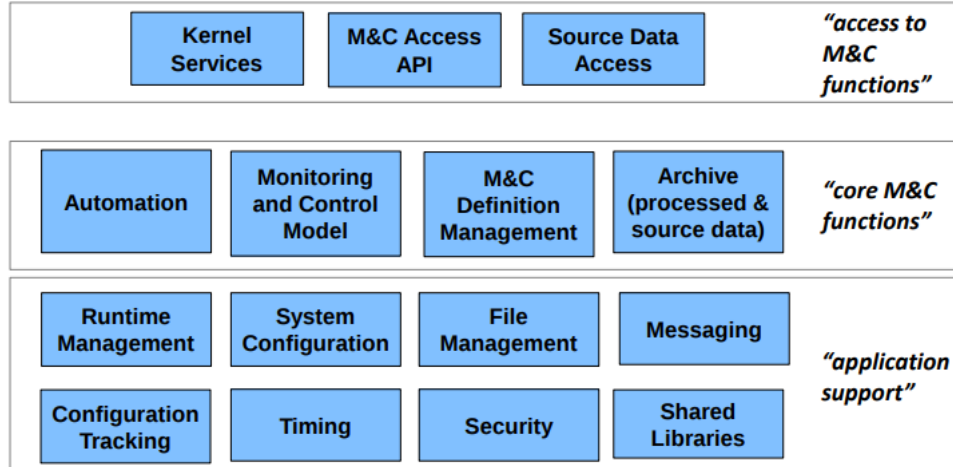


SOURCE: Pecchioli and Carranza (2013)

control functionality in a manner that is independent of the interfaces and the packaging of the input-output data, in Figure 3.2 shows components of the EGS-CC Kernel. The Reference Implementations consist of sets of components which provide functionality required to adapt the Kernel to a specific environment. These components may be replaced without any impact on other components. The Reference Test Facilities is a test environment used for validation of the EGS-CC product. It

contains the external elements for a full validation of the kernel and the reference implementations. The development of the EGS-CC require the application of sev-

Figure 3.2 - Decomposition of the EGS-CC Kernel



SOURCE: Goetzelmann et al. (2014)

eral technologies. Technology plays a key role and a number of different technology have been considered as part of the EGS-CC. As result of the technology selection a simplified overview of the technology suite is shown in Figure 3.3.

The EGC-CC development lifecycle is divided into phases. According to Pecchioli and Carranza (2013), Pecchioli and Carranza (2017) The Phase A(Definition of System concepts and Requirements baseline) has been recently completed, Phase B(Architectural design and Technology baseline definition) just started, Phase C/D (Development)is expected to deliver until the end of 2019 and Phase of Integration and Adoption to be defined.

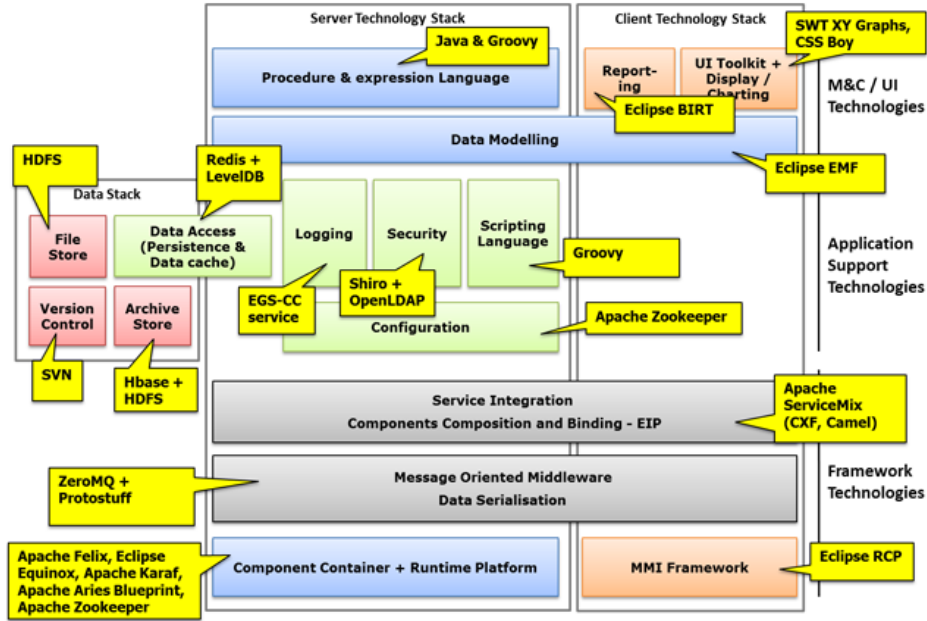
### 3.2.2 Control Channel Toolkit (CCT)

The CCT is a software asset base for a software product line of ground-based spacecraft command and control systems commissioned by United States National Reconnaissance Office (NRO)<sup>4</sup> and built under contract by Raytheon Company<sup>5</sup>. The NRO designs, builds, and operates defense reconnaissance satellites. The asset base

<sup>4</sup><http://www.nro.gov>

<sup>5</sup><https://www.raytheon.com>

Figure 3.3 - Overview of technology suite - EGC-CC.



SOURCE: EUROPEAN SPACE AGENCY (ESA) (2017a)

consists of the different artifacts such as generalized requirements, a software architecture, a development environment definition, a set of reusable software components, and a guide for reusing the architecture and components (CLEMENTS et al., 2001).

The CCT architecture is organized in two primary subsystems: *planning* and *execution*. These subsystems are organized by component categories and each category has its components. Component categories across these two subsystems did not interact(except by shared files). Components of a subsystem might use each other and system functions are accomplished through the operating and interaction of components across different categories. The basis for the architecture's intercomponent communication infrastructure was accomplished with Common Object Request Broker Architecture (CORBA)<sup>6</sup>. Table 3.1 shows this architectural organization.

The execution subsystem referred to the time-critical component categories. These components are responsible for the operations of communication directly with the satellites, examples of such operations are telemetry and telecommand. Figure 3.4 shows the data flow view of CCT execution architecture. The planning architecture

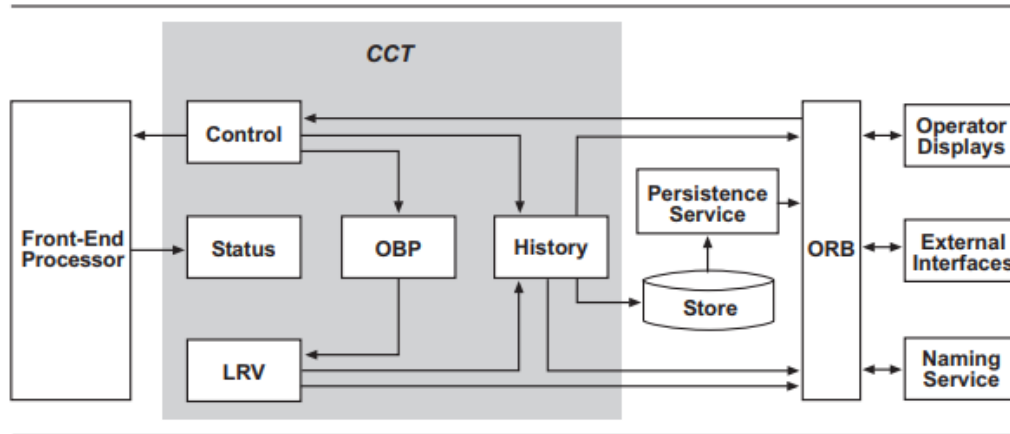
<sup>6</sup><http://www.corba.org>

Table 3.1 - CCT-Subsystems, Categories and Components

Subsystem	Component Category	Components
Execution	Status	4
	Last recorded values	4
	Control	4
	On-board exec	3
	History	5
	Other (Playback)	1
Planning	Orbit	7
	Attitude	4
	Maneuver	5
	Vehicle	3
	Schedule	3
	Evaluation	4
Object services	Object services	10
Infrastructure	Infrastructure	42

SOURCE: Clements et al. (2001)

Figure 3.4 - CCT Execution Architecture: Data Flow View.

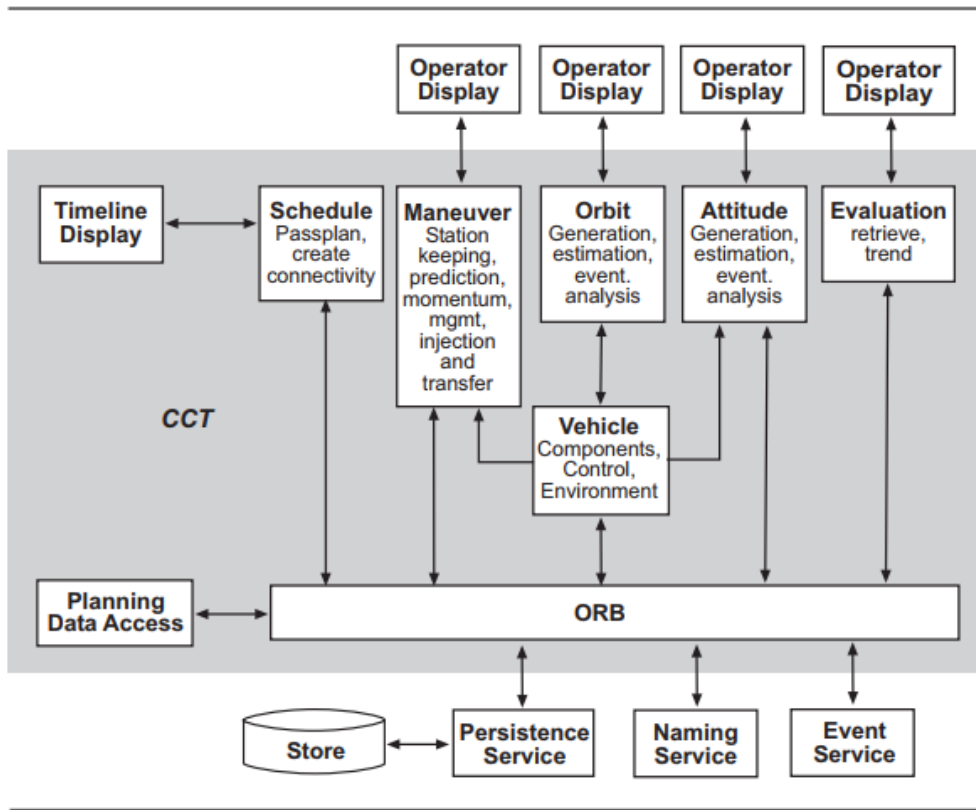


SOURCE: Clements et al. (2001)

referred to the non-time-critical component categories. These components within the six categories provide common features for producing commands which will be sent to the satellites via execution subsystem. Figure 3.5 shows the data flow view of CCT planning architecture.

The execution and planning architecture underwent explicit evaluation led by the

Figure 3.5 - CCT Planning Architecture: Data Flow View.



SOURCE: Clements et al. (2001)

Software Engineering Institute (SEI)<sup>7</sup> using the Software Architecture Analysis Method (SAAM) (BASS et al., 2003). The CCT program provided components to support a real system, the Spacecraft C2 System. Among the various advantages of using the CCT in this business case, are increasing quality and cost reduction and development time. The development costs saved 18.2% of the total forecast and 27.8% savings related to maintenance costs(CLEMENTS et al., 2001).

### 3.2.3 Hifly

The Hifly is a complete commercial solution for SCS, developed by the company GMV<sup>8</sup>. The system is based on COTS and it is compatible with multi-mission and multi-user. Its design is based on a client-server and three-tier architecture (MOREL et al., 2014). The main requirements for architectural design are scalability

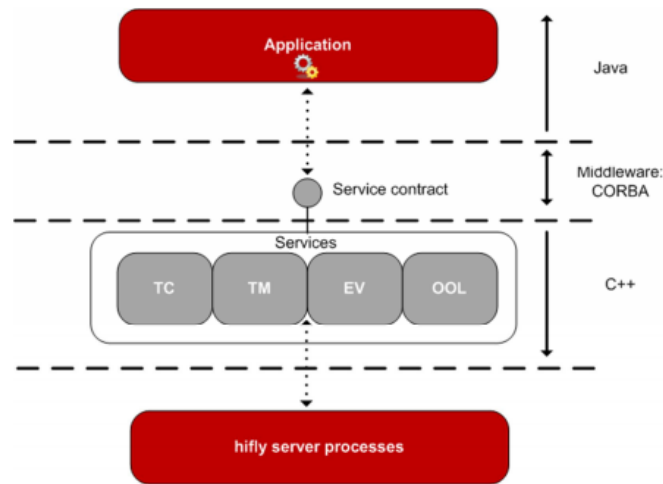
<sup>7</sup><https://www.sei.cmu.edu>

<sup>8</sup><https://www.gmv.com/en/Products/hifly>

and interoperability, to achieve these requirements it was necessary to develop a division between *presentation layers* and *business logic*.

Hifly adopts Service Oriented Architecture (SOA) to execute business logic. The Simple Object Access Protocol (SOAP) layer provides data through the HTTP protocol, which is accessed through Remote Procedure Calls (RPC) with a middleware Corba (LOPEZ; FRAGA, 2012). Any application, exchange data with hifly services layer using CORBA. The service contract, acting as a façade, translate queries and gather data from hifly server. The illustration of this process is shown in Figure 3.6.

Figure 3.6 - Hifly SOA Layer.



SOURCE: Lopez and Fraga (2012)

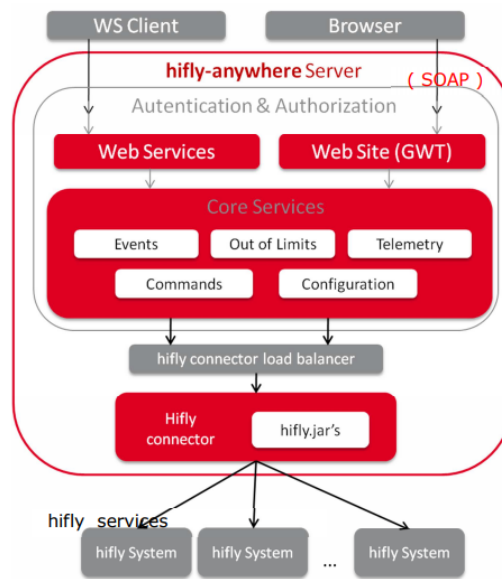
Applications use SOAP to exchange data with an authentication layer for accessing the different web services (Core Services), which translate SOAP requests to the hifly services through of hifly connector. Figure 3.7 shows the hifly anywhere SOAP services logical model.

Hifly is built through a set of components that can be tailored to the needs of the user (GMV, 2017). Next, we present a detailed description of these components.

High-level components:

- a) hifly®: Core fleet monitoring and control system for commercial satellite operations, allows operators to focus on the operations of a specific satellite,

Figure 3.7 - Hifly anywhere SOAP services logical model.



SOURCE: Lopez and Fraga (2012)

while keeping track of events across the satellite fleet. The core incorporates server and client components as follows:

- hifly Server: multimission telemetry and telecommand core processing chains.
- hifly Client: workstation desktop providing operations support either for a single satellite or for a fleet of heterogeneous satellites.

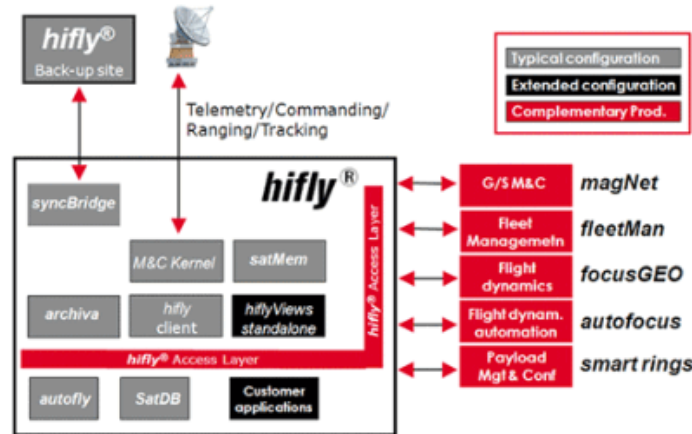
High-level components provide enhanced/high-end capabilities:

- a) archiva: is a processed spacecraft telemetry database to store information and statistical data over different time ranges. It delivers access to telemetry over the whole satellite lifetime, trending analysis and statistical data to spacecraft analysts, engineers and operators.
- b) SatDB: offline Satellite database management system, manages the satellite database by editing and checking consistency of the data, and distributing it to the processing chains for operations.
- c) SatMem: Advanced satellite memory management tool.

- d) SyncBridge: synchronizes operational data between various sites. It can also be used to transfer operational data to non-operational environments.
- e) FleetMan: provides extended fleet management capabilities to satellite control centre deployments.
- f) Views: is an open, advanced telemetry visualization client. It provides flexible telemetry access and highly configurable data presentation.
- g) Autofly: provides procedure automation functionality required to execute and schedule operational procedures for satellite operations.
- h) elBook: is an electronic operations logbook, provides centralized handling of all system incidences observed by end-users.
- i) E2EE: is an end-to-end emulator, simulates space system elements (satellites, ground stations, SCCs, etc).

The following figure 3.8 provides an overview of the above-identified components with the GMV products: focusGEO, magNet, smart rings, autofocus.

Figure 3.8 - Overview of Hifly components and interfaces with GMV products.



SOURCE: GMV (2017)

The use of the Hifly system is indicated for deploying in VMWare<sup>9</sup> virtual environment, providing significant benefits compared to traditional deployments (MOREL et al., 2014).

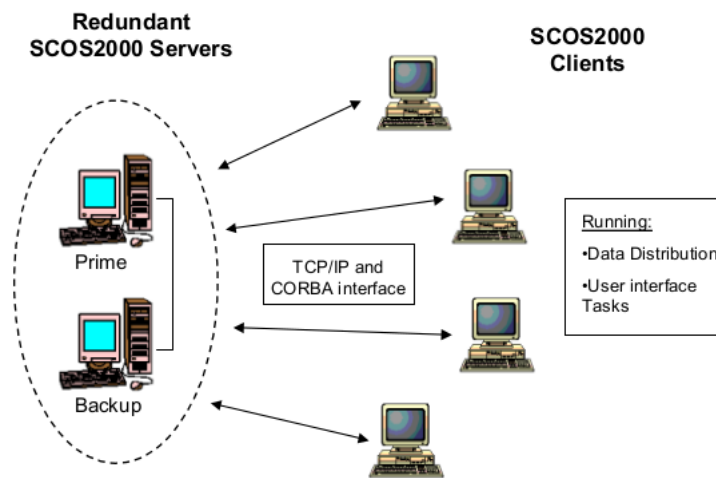
<sup>9</sup><https://www.vmware.com>



### 3.2.4 Satellite Control and Operation System 2000 (SCOS-2000)

The SCOS 2000 is a generic configurable spacecraft monitoring and control system, developed by the European Space Operations Center (ESOC), one of the main ESA centers. The design of SCOS-2000 consists of a set of independent building blocks or subsystems, developed from client-server architecture (NOGUERO et al., 2005; OSORIO et al., 2006), as shown in Figure 3.9.

Figure 3.9 - SCOS-2000 client/server architecture.



SOURCE: Osorio et al. (2006)

The SCOS-2000 concentrates in generic features for all missions, so is developed the concept of the mission family kernel. This kernel is a first level of customization of the generic kernel for supporting a group of missions with similar operational requirements. The objective is the production of mission family kernels for each group of missions, for example Earth observation missions, interplanetary missions, and observatory missions (NOGUERO et al., 2005). SCOS-2000 provides a library of components and services that may be extended and customized to meet specific requirements. The SCOS-2000 kernel is composed of the following subsystems:

- a) Telemetry: responsible for the reception, extraction, modeling and display of telemetry data. It provides a generic monitoring facility;
- b) Telecommanding: responsible for generating, modeling and release of commands;

- c) Online Database: responsible for maintaining the online static mission information base, it contains the definitions of all static elements needed by SCOS-2000 for run-time management;
- d) Data Archiving & Distribution: responsible for filing, retrieval and distribution of data generated (telemetry, telecommands and events);
- e) User Management: responsible by access control to the system functions, and the management of privileges;
- f) Events & Action: responsible for providing event message generation, tracing and handling;
- g) Onboard Software Maintenance: responsible for providing of memory images and memory uplink data;
- h) External Interfaces: responsible for supporting the provision and reception of data to/from external systems.

A common CORBA interface is used throughout components to make the process of TCP/IP connection established between clients and servers. CORBA is a basic middleware for sharing common configuration. It uses a name service to enable server processes to register their location and client processes to retrieve the location in order to connect to the servers (OSORIO et al., 2006).

### 3.2.5 Goddard Mission Services Evolution Center (GMSEC)

The GMSEC is a satellite command and control system. It is designed and defined by NASA's Goddard Space Flight Center (GSFC)<sup>10</sup> to support its satellite missions. The purpose of GMSEC is to simplify integration, allow increased automation, facilitate new adaptations to flight operations components (e.g. Command and Control, Flight Dynamics, Mission Planning, Automation, etc.) with minimum impact to the overall system (CHAMOUN et al., 2006; MAYORGA, 2006; GUDMUNDSSON et al., 2015).

The GMSEC system was developed with the following architectural principles (SMITH et al., 2006):

- a) Standardize interfaces: GMSEC prioritizes the standardization of interfaces - not components. It encourages the access to a broad range of technologies by standardizing interfaces;

---

<sup>10</sup><https://gmsec.gsfc.nasa.gov>

- b) Middleware infrastructure: GMSEC provides a message oriented middleware (MOM) at the center of the architecture;
- c) User choice: GMSEC doesn't decide which components are best nor dictate which components a mission must use. The architecture allows the user or mission to select the most appropriate products based on functional need or personal preference and easily integrate them into a ground system;
- d) General-purpose approach with flight-ground capabilities: The architecture itself should be designed to be adaptable to any number of missions.

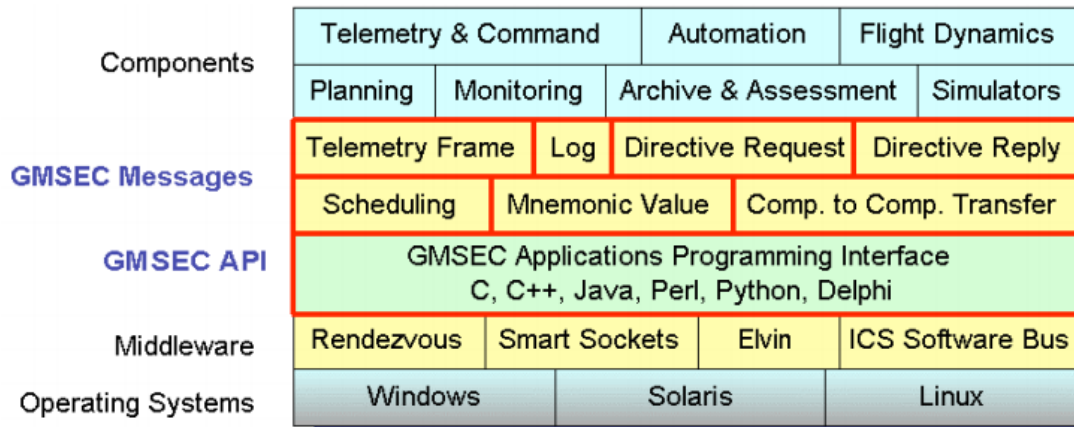
The GMSEC architecture is based on middleware, message-oriented communications and implemented as the message bus. The message-oriented middleware keeps track of the locations of the software components so hard-code node routing is not needed and logical/physical location transitions can occur instantly in the case of failovers. The message bus provides publish/subscribe message passing mechanisms. Applications “publish” messages to the bus. Each message contains a subject name and the standard message contents.

The GMSEC API provides the generalized interface between applications programs and the system middleware. Multiple languages, middleware products, platforms and operating systems are supported. In addition, the API normalizes the behavior of the middleware while allowing access to special functions or capabilities of individual middleware products, in this way, changes in the middleware product does not require changes to the software components. The Figure 3.10 shows an overview of the GMSEC layered architecture.

Some other components or variations that could be part of a component above or even its own subsystem are ([HANDY, 2016](#)):

- Maneuver (prediction, control, analysis, product generation);
- Data Processing;
- Data Analysis and Modeling;
- Data Archive and Data Management;
- Product Generation;
- Data Distribution;

Figure 3.10 - GMSEC Layered Architecture.



SOURCE: Smith et al. (2006)

- Communications and Data Transport;
- Configuration Monitoring and Control;
- Sensor, Instrument and Payload Operations.

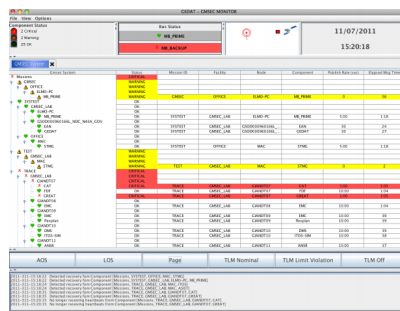
Next, is described an overview of the main GMSEC in-house components and how they integrate with the GMSEC framework to provide a versatile platform:

- GMSEC Environment Diagnostic Analysis Tool (GEDAT): provides a graphical display of the GMSEC environment, identifying and tracking all GMSEC-compliant components connected to the bus. It alerts the user to various error conditions including component and bus failover, as presented in Figure 3.11. GEDAT also collects and shows computational resources of each node(CPU, memory, network, disk), as presented in Figure 3.12;
- System Agent (SA): provides health information about the computer hosting the agent to other GMSEC components. These agents provide some of the raw data used by GEDAT. This includes host identifying information, CPU utilization, memory utilization, and disk utilization;
- Criteria Action Table (CAT): is a rules-based automation tool, with purpose of monitoring real time system messages and certain specific events can be detected in order to take associated predefined actions in an effort

to automate and enhance the reliability of the system. CAT automates a wide set of ground system functions that otherwise might require dedicated staff.

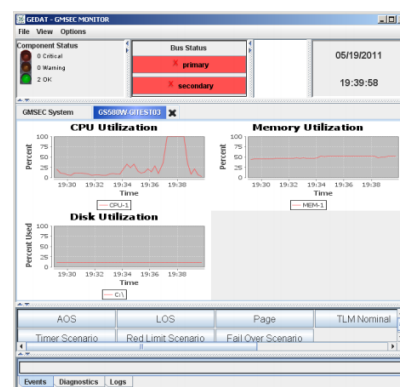
- d) GMSEC Reusable Event Analysis Toolkit (GREAT): is a comprehensive system for event/log messages. It displays all messages transmitted along the information bus, message archive & retrieval, and data file format conversion.
- e) GMSEC Remote Access Service Provider (GRASP): provides the capability for users to access remote viewing, via the Internet, of GMSEC information. GRASP addresses the transfer of information from a GMSEC environment to a web server.

Figure 3.11 - Screen Shot of the GEDAT.



SOURCE: Handy (2016)

Figure 3.12 - GEDAT - resource information (CPU, memory, network, disk utilization).



SOURCE: Handy (2016)

The GMSEC and the Hifly systems performed an interoperability process, that involved the development of an adaptor that would bridge the different architectures of GMSEC (message-based middleware) and Hifly (CORBA-based API). In addition, there were a data adaptation of a typical NASA database and the adaptation of the telemetry message and event flows (CHAMOUN et al., 2006).

Another example of system integration was performed by Aerospace Ground Systems Laboratory in four satellites telemetry, tracking and command systems. This integration occurred through the use of adaptors developed by vendors and by Ground

Systems Laboratory (SULLIVAN et al., 2009).

### 3.2.6 Global Educational Network for Satellite Operations (Genso)

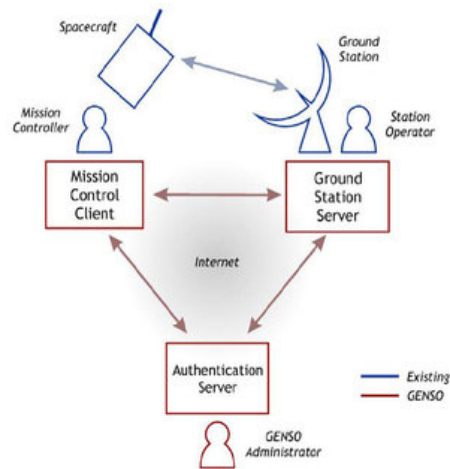
The Genso is an ESA initiative collaborating with the International Space Education Board (ISEB), with the participation of several universities, such as University of Tokyo, Vienna University of Technology, Cal Poly State University, Jean Monnet University (LEVEQUE et al., 2007). It was officially created in October 2006 with its main objective being the provision of communications to university satellites and, moreover, to create an international network of university ground stations. The objective of the project is to train students in space system and increase the level of access to orbital spacecraft through the sharing of earth stations (CASTRO et al., 2012). Among other GENSO's objectives are the following (EUROPEAN SPACE AGENCY (ESA), 2017b):

- Global access to mission operators of educational and amateur radio spacecraft.
- Remote access for operators to real-time mission data, even in cases when their local ground station is experiencing technical difficulties.
- Scheduling of uplinks through ground stations.
- Definition of standard solution for the educational ground-segment hardware, designed to optimize GENSO's performance at minimal cost.
- Close collaboration with amateur radio to support a common interface in order to apply for frequency allocation and coordination.

In the Genso network, satellite operators can control their satellite remotely from any shared earth station in the network, regardless of their geographic location. Communication between operators and stations is done through the Internet, as well as the main elements of the Genso architecture that are deployed in distant locations geographically.

Genso network has three main elements in its architecture: Authentication Server (AUS), single point of authentication in the network; Ground Station Server (GSS), which is deployed at earth stations; and Mission Control Client (MCC) used by satellite operators (LEVEQUE et al., 2007). Figure 3.13 presents an overview of the elements of Genso's architecture.

Figure 3.13 - Architecture of the GENSO network.



SOURCE: EUROPEAN SPACE AGENCY (ESA) (2017b)

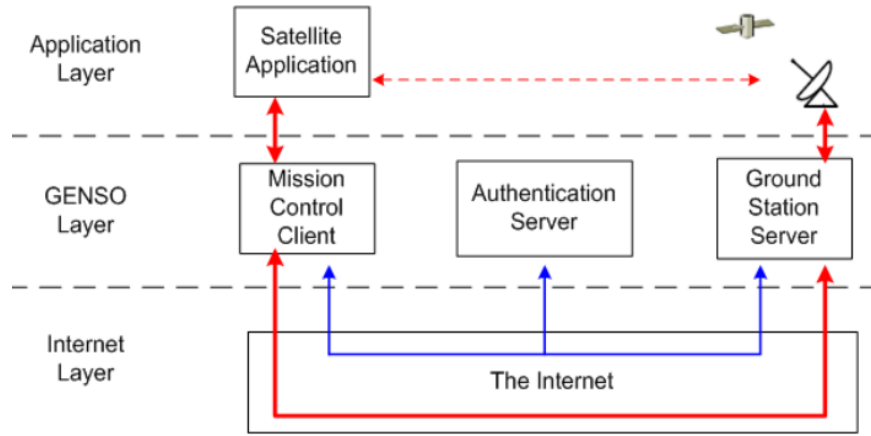
The AUS enables MCC and GSS to participate in the network, performs all communication measurement, acts as a centralized authentication server. This server is located at the University of Vigo, with a contingency at CalPoly. These servers are also synchronized over the Internet in order to avoid failure on any particular node.

The GSS is the software that controls the hardware of the stations, such as the rotors, TNC and the frequency settings of the radio. GSS must be installed on all ground stations and connected to the AUS. The GSS application is also used in automated remotes, scheduling negotiations, and booking with the MCC for specific missions.

MCC is used by satellite operators to schedule passage and receive data from earth stations by GSS. For each controlled satellite will be necessary to install an MCC instance. When the MCC is authenticated in AUS, it will receive a list of available earth stations with their configuration parameters. The GSS and MCC communicate with one another in a Peer-to-Peer (P2P) fashion. Thus, the GENSO architecture is a hybrid between a P2P network and a centralized network. Figure 2.11 presents an overview of the Genso layered architecture. The red solid lines show the data path for data downloaded from and uploaded to the spacecraft. The blue solid lines show the control data path. The red provides a virtual link to the spacecraft (LEVEQUE et al., 2007).

The Genso's architecture is scalable to several stations, its implementation adopts

Figure 3.14 - Genso layered architecture.



SOURCE: LEVEQUE et al. (2007)

open source software, with development in Java language and part developed in C# and C language. It adopts the COTS components approach and certificates and HTTPS are used for communication security.

### 3.2.7 Satellite Network (SatNet)

The SatNet is an initiative of Cal Poly University together with Fundación Barrié of Spain, currently the University of Vigo (Spain) and INPE (Brazil) also participate in this project. The project aims to develop reusable software to control CubeSats from shared earth stations. In the sharing of ground stations, each earth station contributes with its resources and capacity of communication, promoting a set of new communication channels of different locations. Thus, increased ground segment coverage and providing control and tracking of small satellites for a longer time. It uses an approach based on heterogeneity, allowing the integration of different earth stations into a single system, and provides a distributed solution based on cloud computing. The project provides a network infrastructure that allows operators to remotely access ground stations (PARDAVILA et al., 2014).

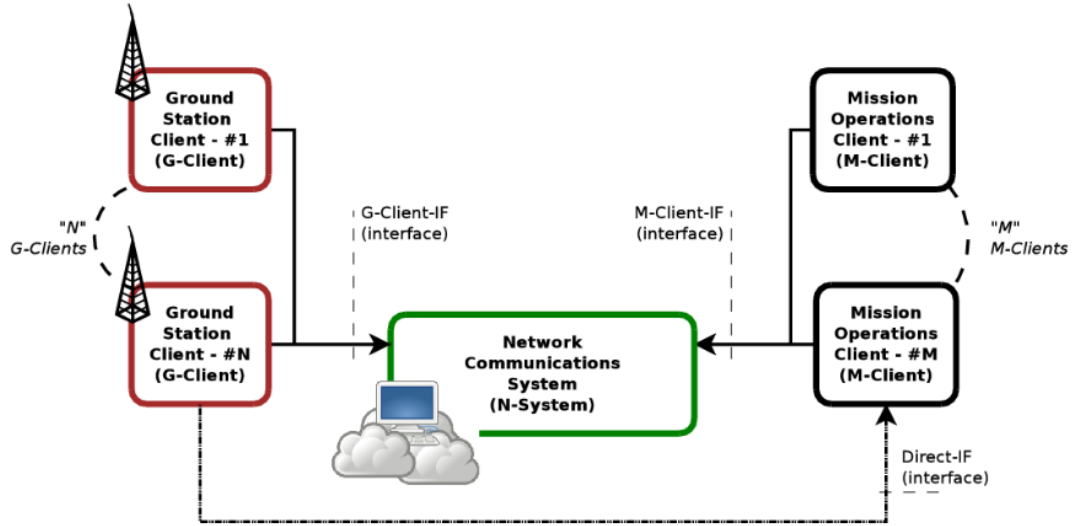
#### 3.2.7.1 SatNet Architecture

The SatNet architecture is composed of systems and communication interfaces, the integration of these systems provide distributed and orchestrated functionality. The main systems that make up the system's architectural solution are the *Ground Station Clients*, *Network Communications System* and *Mission Operations Clients*. Fig-



Figure 3.15 presents an overview of the SatNet architecture and its systems and interfaces.

Figure 3.15 - SatNet architecture overview.



SOURCE: Pardavila (2013)

The elements of the SatNet architecture are shown briefly in Figure 3.15 and defined as follows (PARDAVILA et al., 2016):

- A set of software clients for spacecraft operators to command remotely the satellites. From now on, they will be defined as Mission Operations Clients or M-Clients for short.
- A set of software clients for providing direct access to the services of the ground station facilities. From now on, Ground Station Clients or G-Clients for short.
- A cloud system for the coordination of the communications in between these two types of clients. From now on, Network Communications System or N-System. It is important to note that the N-System is not a single server but a cloud-computing-based system. This way and depending on further implementation decisions, this cloud system may evolve into a network of interconnected servers that will provide the service required.

The N-System implements the following interfaces for permitting an automatic communication among software entities, without the need of direct human interaction:

- G-Client Interface (G-Client-IF), that permits the ground station clients to connect to the network services.
- M-Client Interface (M-Client-IF), that permits the mission operations clients to connect to the network services.
- Direct Client Interface (Direct-IF), that permits the mission operations clients to connect directly to the ground station clients.

The current implementation of the SatNet network provides only the M-Client-IF and the G-Client-IF interfaces, being the Direct-IF left for future releases (PARDAVILA et al., 2016). It provides all source code needed to connect ground stations and control centers under apache version 2.0 license <sup>11</sup>. On the website <sup>12</sup> it's found the user interface that gives access and control to the network.

### 3.2.7.2 SatNet Services

SatNet is based on services, these services are made available to meet the main functionalities and the integration between the systems. SatNet provides the following services (classified in accordance with three categories) (PARDAVILA, 2013):

- Management Services Category
  - Registration Service.
  - Configuration Service.
  - Information Service.
- Scheduling Services Category
  - Assisted Scheduling Service.
  - Private Scheduling Service.
- Communication Services Category
  - Assisted Communications Service.

---

<sup>11</sup><https://github.com/SatNet-project>

<sup>12</sup><https://SatNet.aero.calpoly.edu>

- Private Communications Service.
- Non-Scheduled Communications Service.

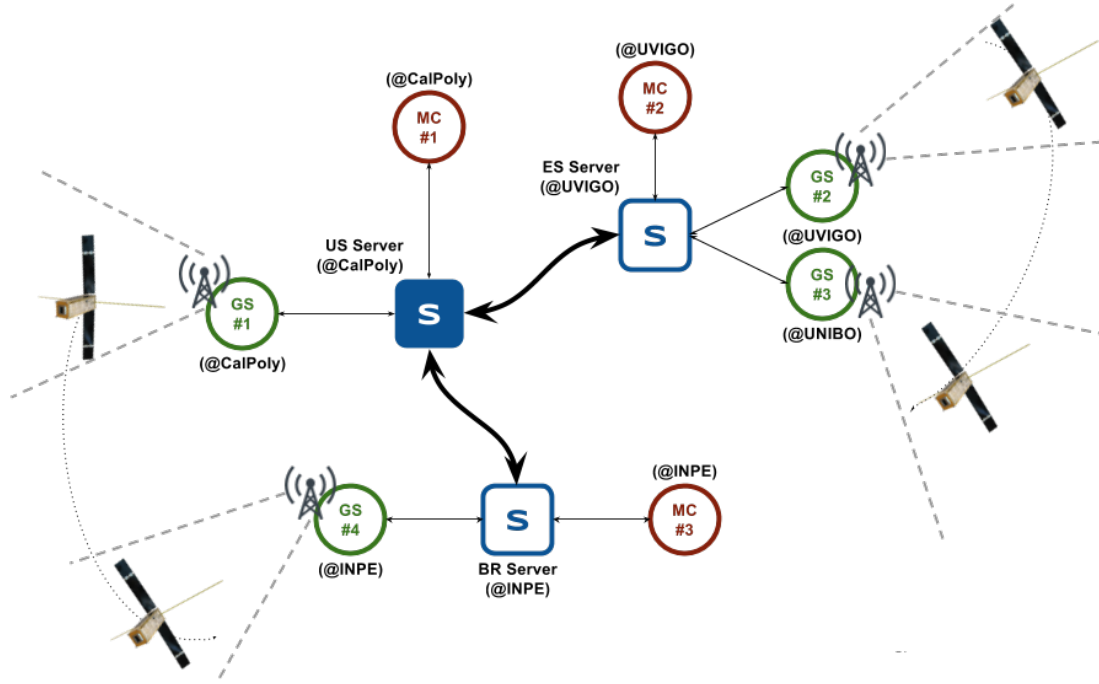
The SatNet network is designed to give a solution based on flexible computing in the cloud. This solution allows the use of its resources in the cloud so that the CubeSat type ground stations can have access.

### **3.2.7.3 SatNet Integration**

SatNet still in development allows having a global ground station network, with global coverage of the satellites willing to be a part of the network. SatNet currently has an increasing number of integration of earth stations, the United States and Spain are countries that are part of the project.

At present, some trial activities are taking place for the integration of the ground station INPE in the SatNet network. The envisaged scenario is shown in Figure 3.16, where the current SatNet deployed network server is located at CalPoly where it is already providing operational services. In the near future, two more servers are planned to be deployed, namely: one at Vigo in Spain and the one prospectively located at INPE so that it will provide coverage for the southern hemisphere (PARDAVILA et al., 2016).

Figure 3.16 - SatNet integration.



SOURCE: Pardavila et al. (2016)

The integration process follows a list of steps and defines some keys parameters for the new integrating ground station node to the main SatNet network (PARDAVILA et al., 2014) : (i) Set new ground station location, (ii) Define new ground station metadata, (iii) Define channel information for the new ground station and, (iv) Configure network services for additional client node.

## 4 ESTABLISHMENT OF A REFERENCE ARCHITECTURE FOR SATELLITES CONTROL SYSTEM

This chapter reports on the establishment of Reference Architecture for SCS (SCS-RA). This reference architecture is aligned with the ProSA-RA process. We adopted the systematic process ProSA-RA (NAKAGAWA et al., 2014) to establish the SCS-RA, considering that it prioritizes the design, representation, and evaluation of reference architectures and has been already applied in the establishment of many reference architectures (NAKAGAWA et al., 2014). The establishment of SCS-RA involved the development of different steps and according to ProSA-RA process. Next, we present a detailed description of these steps and activities.

The organization of this chapter follows the steps of ProSA-RA. Section 4.1 details the sources of information used to establish SCS-RA. Section 4.2 reports on the requirements of the reference architecture elicited from these information sources. Section 4.3 describes architectural views of SCS-RA. Finally, Section 4.4 presents a architectural evaluation of SCS-RA.

### 4.1 Step RA-1: Information Source Investigation

In this first step, two main groups of information sources related to satellites which control activities were identified: (i) CCSDS recommendations and reports; (ii) ECSS standards. These groups are initiatives established to develop a coherent set of standards for use in all space activities. The following is a summary of the main guidelines considered in this stage:

- a) ECSS-E-ST-70C: It has some principles and requirements to apply in the engineering of the ground segment and in the mission operations. It also presents the general view and the elements of a space system (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2000).
- b) CCSDS-520.1-M-1: It is a recommended practice which defines a reference model for the mission operations. It provides a common base to support the development of the CCSDS' recommended standards (which specify the mission service operations). It also provides a reference to keep consistency of these standards (CCSDS, 2010a).
- c) CCSDS-520.0-G-3: It presents a general view of the concepts of mission service operations. It also defines a framework of mission service operations to use for monitoring and controlling the satellites (CCSDS, 2010b).

- d) CCSDS-521.1-R-1: It defines in an abstract way the common services to monitor and control the satellites. These services offer support to mission operations present in most systems. This standard defines specifications of the interfaces of these services as to allow interoperability among agencies through the installed software. It also allows the interoperability where the collaborating agencies may share resources and obtain advantages of shared installations (CCSDS, 2009).
- e) CCSDS-522.0-R-2: It contains a high level formal specification to define a model of a common object. The common object is applied to the mission service operations defined in (CCSDS, 2010b). The specification defines a standard for the services of the common object model and a standard for the support services for the model of these objects (CCSDS, 2008).
- f) CCSDS-311.0-M-1: It aims at providing a standardized approach to high level description of architectures and a pool of conventions for space data systems. It defines concepts; it presents advantages and examples of architectural views. Among the objectives there is the goal to establish a general view of the CCSD's recommended methodologies, as to define and develop the reference architecture to the domain. It was inspired by the model RM-ODP and is compatible with the best software architecture practices available at ANSI/IEEE 1471-2000 (INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2000). This recommendation presents concepts and conventions representing the perspectives of: business, functional, connectivity, communication and information views (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008).
- g) ECSS-E-ST-40: It encompasses the software engineering aspects of a space system. It aims at contributing to formulate the requirements of the space environment. It presents a general view of engineering of the requirements for the whole life cycle of the space systems projects (ECSS, 2009a).

Hence, SCS-RA should be consistent with these initiatives. Thus, the guidelines and their contents that contribute to the definition of the architectural requirements were first identified. An example of this is the API specification for cross support defined in (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2005), so the SCS-RA should be compatible with this specification. Table 4.1 shows the reference and contents of interest captured for the definition of SCS-RA.

Table 4.1 - Contents of interest captured for the definition of SCS-RA

Reference	Contents of Interest
ECSS-E-ST-70C (EUROPEAN CO- OPERATION FOR SPACE STANDARD- IZATION (ECSS), 2000)	Ground Station Elements, decomposition, communication.
ECSS-E-ST-40C (ECSS, 2009a)	Requirements, software, process, software engineering.
ECSS-E-ST-70- 31C (EUROPEAN COOPERATION FOR SPACE STAN- DARDIZATION (ECSS), 2008)	Integration, monitoring and control data, data standards, activities, events.
CCSDS 910.4-B (THE CONSULTA- TIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2005)	Cross support, SLE API, proxy, logical view, gateway.
CCSDS-520.1-M (CCSDS, 2010a)	Operation, telecommand, decomposition, quality of services, architecture, domain components, portability, interoperability, security.
CCSDS-521.1-R (CCSDS, 2009)	Common services, inter-agency interoperability, operations model, service directory.
CCSDS-522.0-R (CCSDS, 2008)	Operations, objects, entity, structure, service recovery, gateway, composition.
CCSDS-520.0-G (CCSDS, 2010b)	Operations, telecommand, telemetry, components, interoperability, API, independent platform, location service, schedule.
CCSDS-311.0-M (THE CONSULTA- TIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008)	Architecture, reference model, conventions, standardization, distributed system, architectural views, registry.
CCSDS-312.0-G (THE CONSULTA- TIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2013)	Routing of queries, discovery, repository, registry, components software.

SOURCE: Author

Besides these groups of information, other information sources were identified: (i) SCS domain that comprises of researchers and experts in space technology, ground station operators, satellites controllers, and systems engineers from INPE; (ii) RA experts, involving researchers, professors, experienced architects, doctoral students from Institute of Mathematics and Computer Sciences of University at São Paulo (ICMC-USP); and (iii) systems in the domain, including , ESA’s SCOS-2000, GMV’s Hifly, CalPoly’s SatNet and INPE’s SatCS. The mapping of the features these sys-

tems are shown in Table 4.2.

Table 4.2 - Mapping of the Features in Existing Systems

Feature	SCOS-2000	SatNet	Hifly	SatCS
Telemetry	✓	✓	✓	✓
Telecommand	✓	—	✓	✓
Tracking	✓	✓	✓	✓
Flight Dynamics	✓	—	✓	—
Rotors	✓	—	✓	—
Monitoring	✓	✓	✓	✓
Simulation	✓	—	✓	—
Schedule	✓	✓	✓	✓
Ranging	✓	—	✓	—
Attitude	✓	—	✓	—
Spacecraft	✓	✓	✓	✓
Ground Station	✓	✓	✓	✓
Antenna	✓	✓	✓	—
Maneuver	✓	—	✓	—

SOURCE: Author

## 4.2 Step RA-2: Architectural Analysis

In this step, we analyzed every information extracted from the last step and requirements for our RA were derived from them. In addition, the SCS-RA components have been identified and defined.

### 4.2.1 Architectural Requirements of SCS Domain

The set of requirements was defined according to two groups: Architectural Domain Requirements (ADR) and Technical Architectural Requirements (TAR). In total, 13 ADR and 8 TAR were established and are shown in Table 4.3.

Table 4.3 - SCS-RA Requirements

ID	Description
<b>Architectural Domain Requirements</b>	

Continued on next page



**Table 4.3 – SCS-RA Requirements (cont.).**

<b>ID</b>	<b>Description</b>
ARD-01	The Reference Architecture must enable the development of SCS to different types of mission and multi-mission.
ARD-02	The Reference Architecture must enable the development of SCS that share ground segment resources.
ARD-03	The Reference Architecture must enable the development of SCS to different types of satellite mass categories, including nanosatellites, microsatellites, nanosatellites, picosatellites, and femtosatellites.
ARD-04	The Reference Architecture must enable the development of SCS to control several satellites, constellations or clusters.
ARD-05	The Reference Architecture must enable the development of SCS that provides and executes operations in different SCS.
ARD-06	The Reference Architecture must enable the development of SCS to control satellites in any orbit, such as: Low Earth Orbit (LEO); Medium Earth orbit (MEO); High Earth Orbit (HEO); and Geo-stationary Orbit.
ARD-07	The Reference Architecture must enable the development of SCS able to store, share, and distribute data acquired from platforms and payloads.
ARD-08	The Reference Architecture must enable the development of SCS to different organizational structures.
ARD-09	The Reference Architecture must enable the development of SCS that monitor the health status of ground station resources and satellites.
ARD-10	The Reference Architecture must enable the development of SCS that supports the exchange of data standardized by the domain.
ARD-11	The Reference Architecture must enable the development of SCS with the frequency bands of communication, such as, UHF, VHF and Band S.
ARD-12	The Reference Architecture must enable the development of SCS that can coordinate, monitor, and acquire data from different types of ground resources.
ARD-13	The Reference Architecture must allow the development of SCS that are compliant with applicable laws, regulations, and standards.
<b>Technical Architectural Requirements</b>	

Continued on next page

**Table 4.3 – SCS-RA Requirements (cont.).**

<b>ID</b>	<b>Description</b>
TAR-01	The Reference Architecture must enable the development of SCS through plug-and-play software components.
TAR-02	The Reference Architecture must enable the development of SCS with blocks developed in different programming languages.
TAR-03	The Reference Architecture must enable the development of interoperable SCS.
TAR-04	The Reference Architecture must enable the development of SCS through dynamic architecture.
TAR-05	The Reference Architecture must enable the instantiation of SCS through composition of components.
TAR-06	The Reference Architecture must enable the development of SCS that allow the use of heterogeneous databases.
TAR-07	The Reference Architecture must enable the development of SCS that make scalability easy.
TAR-08	The Reference Architecture must enable the development of SCS with monitoring and management of quality attributes.

SOURCE: Author.

In addition to the requirements mentioned, there are several requirements to be addressed, such as data security, performance and usability. These attributes are needfuls requirements and not specific to a domain, so these requirements are also considered in the development of SCS-RA.

#### **4.2.2 Components of SCS Domain**

A good exercise to comprehend complex domains is to execute and organize its decomposition. Considering the complexities of the SCS's domain and aiming at the division of responsibilities, we observed the need to perform a conceptual decomposition. To perform this decomposition we considered the architectural characteristics and requirements anteriorly presented. The result of this analysis defined the following functional categories:

- a) Coordination category: It is responsible for the discovery and for sharing the resources.
- b) Management category: It is responsible for managing the data flux and for controlling operations.

- c) Business category: It is responsible for the functionalities of the SCS's domain.
- d) Persistence category: It is responsible for the persistence and for recovering the data.

The conceptual decomposition aims at separating the complexity of the domain to identify the components and their categorization. The decomposition of the domain does not refer to the definition of architectural layers or to the modularization of the domain.

Identification of the Domain's components The identification of the components consists in describing the functionalities of each identified component. Table 4.4 presents the previously defined functional categories and their components. The identification and description of SCS components are detailed in the sequence.

Table 4.4 - Category

Category	Components
Business	Telecommand, Telemetry, Tracking, Flight Dynamics, Spacecraft, Ground Station, Rotors, Antenna, Simulation, Monitoring, Schedule, Ranging, Attitude, Maneuver
Management	Gateway, Security, Routing, GUI, Registry
Coordination	Discovery, Compositing, Administrator, Mashup

SOURCE: Author

#### 4.2.2.1 Registry Component

All running components must be registered. In this register, they need to inform their location through an IP address, their access port, and available interfaces. The Registry Component stores this data and provides it to the Discovery Component. The information of the components is dynamic and the Registry Component needs to update these data dynamically.

#### 4.2.2.2 Mashup Component

When organizations are interoperating, some information must be provided in a unified way. The Mashup Component is responsible for researching and providing

this information. The information refers to the satellites, ground stations and the scheduling of operations.

#### **4.2.2.3 Discovery Component**

The SCS-RA components are independent, have execution responsibilities and, work in a collaborative way. These components are registered during its execution and the Discovery Component provides information from these components. The information provided are: physical execution location; its access address; interfaces; etc.

#### **4.2.2.4 Composition Component**

The Composition Component is responsible for instantiating an SCS, which uses a composition process. This process searches and selects components stored in repositories and the computational resources are set up. After the download and initiation of components, the instance of SCS is provided for use.

Administrator Component: This component manages and monitors the computational resources(memory, CPU, and disk) of each component running. Resources consumed by the components are assigned according to the requirements of each component and can be changed at runtime.

#### **4.2.2.5 Orbit Calculator Component**

This component is responsible for projecting orbit parameters for the satellite and its passes over ground stations. These parameters base themselves on the observed data (determination of the state) and then propagate the orbit along time. The determination of the orbit and the project for the positioning of the satellites is done considering the Keplerian elements, such as the angular momentum, orbital inclination, right ascension of the ascending node, eccentricity, argument of perigee, etc.

The Keplerian elements are also included in the TLEs (Two Line Elements)<sup>1</sup>, which correspond to a data format with two lines of 69 digits. With this format (TLE), it is possible to define the orbit trajectory, through the application of formulas and also considering the identification of the space element provided in a defined format according to ([ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2007](#)).

---

<sup>1</sup><https://www.celestrak.com/NORAD/elements/>

#### **4.2.2.6 Telemetry Component**

The Telemetry Component does the functionalities of the platform telemetry and of the payload telemetry. The platform telemetry is stored and decoded to verify the state values of the many subsystems of a satellite. These values are used to monitor and to control the functioning of the satellite. The payload telemetry is received, stored and forwarded to the mission center.

#### **4.2.2.7 Maneuver Component**

The Maneuver Component refers to the process responsible for designing and generating the sequences of maneuvers and it is done sending remote control. These sequences usually use propulsion shots to change the orbit of the satellite. The maneuver is the key to maintain the orbit, the attitude, and the planning activities for the maneuvers depend heavily on the satellite's and the mission's details.

#### **4.2.2.8 Telecommand Component**

The telecommand component consists in the activities of planning, monitoring and sending the remote control to the ground stations. The telecommand is formed by a data structure, which defines a data package. The telecommand has a schedule to execute, a sequence of execution and it may be scheduled in many ground stations.

#### **4.2.2.9 Schedule Component**

The schedule component refers to the scheduling of all the SCS's activities. These activities consist of the scheduling of telemetry, remote control and maneuvers. The scheduling process results in a chronological pool of scheduled activities. Through co-operation among space agencies, the scheduling of activities is done in a distributed and collaborative way.

#### **4.2.2.10 Spacecraft Component**

The spacecraft component contains all the functionalities and the specification of the satellite. These are data such as: unique spacecraft identifier, its subsystems with its operational parameters, mass values, attitude values, orbit values, etc. These values are stored, recovered and afterwards they are compared during the monitoring and controlling process of the satellite.

#### **4.2.2.11 Ground Station Component**

The ground station component has information and functionalities of the ground stations, which do the linkage with the satellites. This information is necessary to do the controlling operations of the satellites. Some relevant data are: the position with the latitude and the longitude; the kind of station (fixed or mobile, tracking station or payload reception station); kinds of transceiver, transmitter, modem, network protocols, etc. The SCS is responsible for the monitoring of this equipment.

#### **4.2.2.12 Tracking Component**

The tracking component presents the satellite's position almost in real-time. The positioning is relative to its global position. The satellite's position is provided through the Orbit Calculation component. The satellite is presented in a visual map with the current position and the tracking of the next positions and passes.

#### **4.2.2.13 Gateway Component**

The Gateway component centralizes the functionalities of security, router and interfaces for the operators, through the composition of these components. These functionalities must be centralized to better the control and the managing.

### **4.3 Step RA-3: Architectural Synthesis**

The architectural synthesis was conducted considering the objectives of the SCS-RA, architectural requirements, and information sources previously identified. SCS-RA is described using architectural views, which are an effective way to describe RA (NAKAGAWA; MALDONADO, 2008; GUESSI et al., 2011). The use of views to describe architectures is defined by ISO/IEC/IEEE 42010 (INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2007) and also used to describe space data systems (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008). During this step, the following views were considered and are described hereafter: enterprise, structural, data flow, logical, and composition.

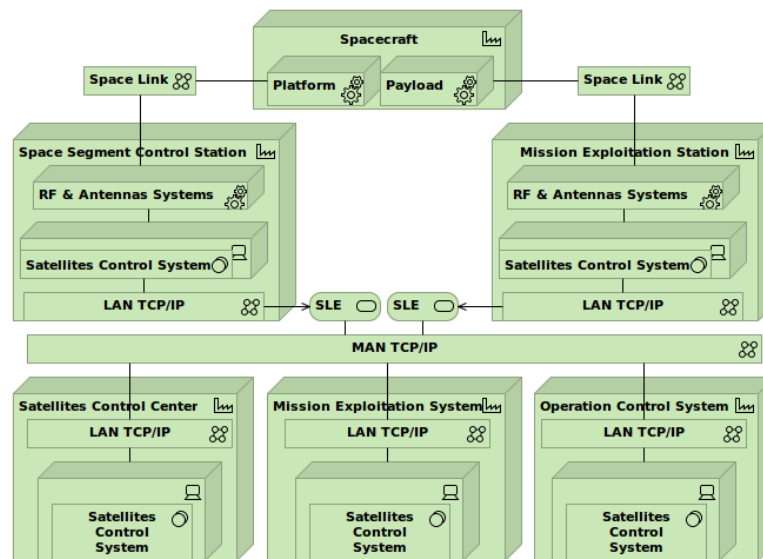
#### **4.3.1 Enterprise View**

The enterprise view is one of the views that has been already used to depict space data systems (THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS, 2008). The ground segment has several forms of organization and involves several constituent systems. Each space agency defines its physical structures (EUROPEAN CO-

OPERATION FOR SPACE STANDARDIZATION (ECSS), 2000) and a RA for SCS must serve different organizational forms. SCS-RA is a distributed and dynamic architecture, which means that SCS are run in several places and have potential to dynamically change their running. This process happens through software components that run independently of any networked computational structure. This means that a SCS is the constitution of one or more software components running in different organizational structures.

This view was designed using ArchiMate, an enterprise architecture modeling language based on IEEE 1471 standard (INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS, 2000) and maintained by Open Group<sup>2</sup>. Figure 4.1 shows how an instance of a SCS based on SCS-RA is deployed in accordance with organizational structures presented by ECSS standard in (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2000).

Figure 4.1 - Enterprise View of SCS-RA in organizational structures ECSS Standard.



SOURCE: Author

#### 4.3.2 Structural View

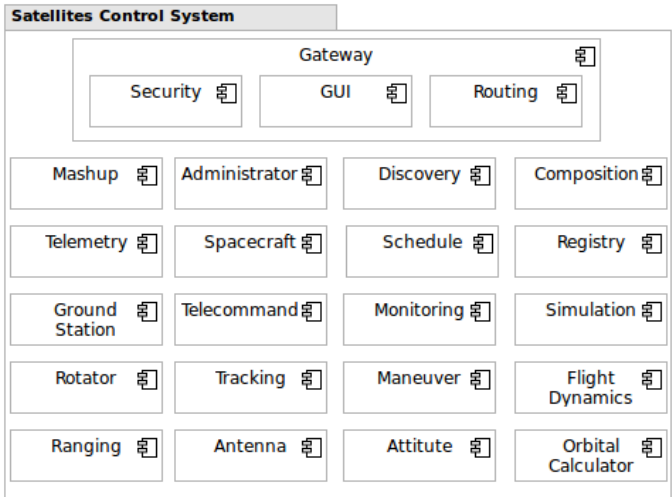
The structural view makes it possible to delimit the context of the RA, as well as its internal elements. The main structures of SCS-RA are components, which run inde-

<sup>2</sup><http://www.opengroup.org>

pendently and could be developed by any programming language. SCS-RA defines all components to serve the satellite control domain. During the instantiation of SCS-RA, each SCS must define the most appropriate components for every mission.

This view was designed using UML components diagram, as shown in Figure 4.2. This view has 21 components that encapsulate the SCS functionalities. In particular, the gateway component consists of three sub-components (Security, GUI, and Routing). Each component has a textual description of their functionalities and APIs for definition of provided and required interfaces.

Figure 4.2 - Structural View of SCS-RA.



SOURCE: Author

### 4.3.3 Data Flow View

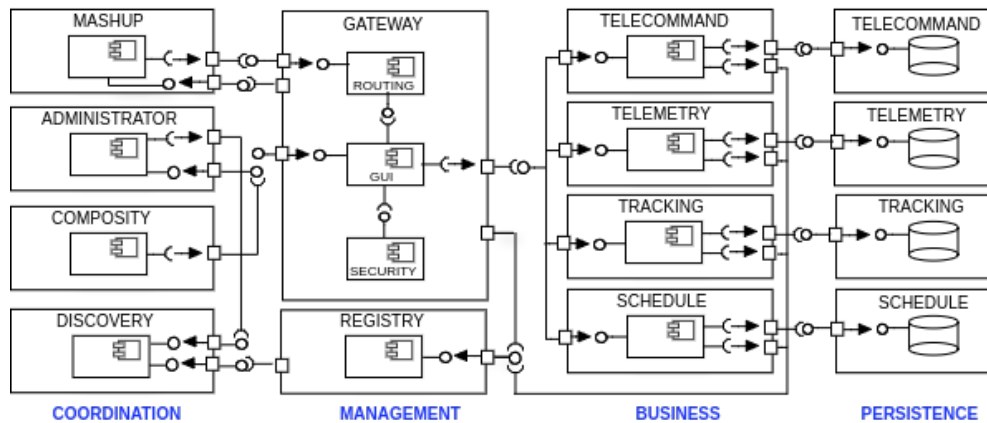
The data flow among the components of SCS is identified in this view and is shown in Figure 4.3, and it was also designed using the UML component diagram. For better understanding, these components are classified into four categories: coordination, management, business, and persistence. For legibility reasons, only telecommand, telemetry, tracking, and schedule components are represented.

### 4.3.4 Logical View

The logical view provides a basis for understanding the structure and organization of SCS-RA, and it is decomposed into components that interoperate through interfaces.



Figure 4.3 - Data Flow View.



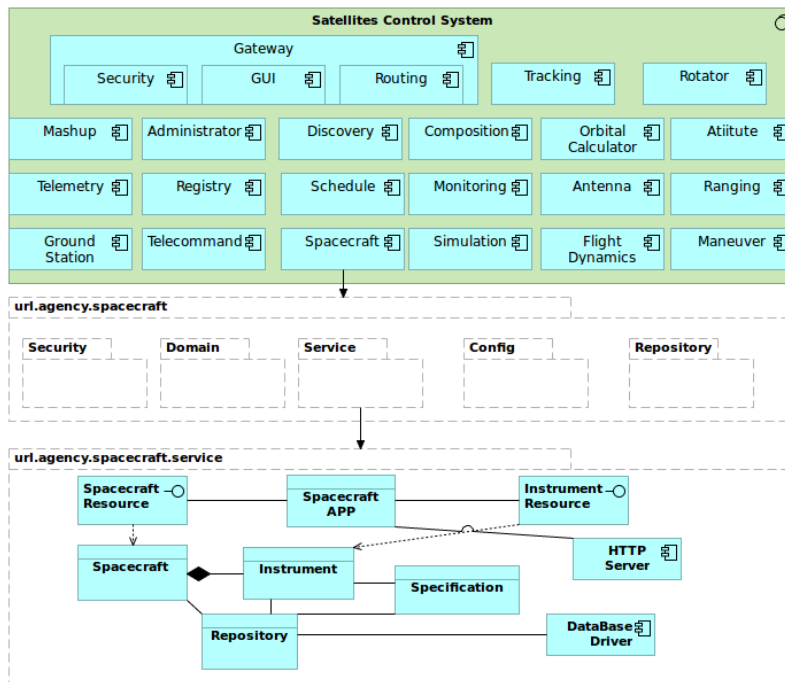
SOURCE: Author

Components encapsulate the SCS functionalities and are decomposed into packages that have several objects. ArchiMate was once again used to design this view and Figure 4.4 shows the components of SCS and in particular the Spacecraft component decomposition in packages and the objects that encompass the service package.

#### 4.3.5 Composition View

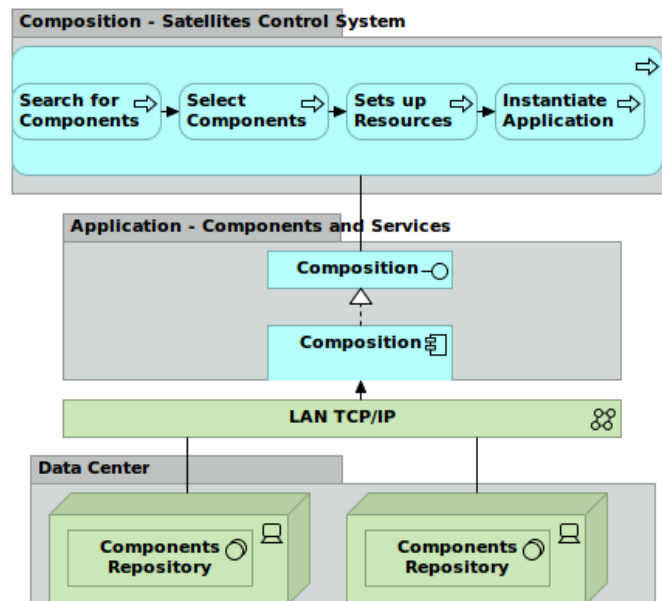
SCS-RA enables the addition, deletion, and exchange of software components that are selected to better meet the operational needs of a particular SCS. The instantiation of an SCS is performed through the components composition process. This process is performed visually, textually or through models by Composition component. This component searches for components stored in repositories, components required for a specified SCS are selected and computational resources are set up. After the download and initiation of components, the instance of SCS is provided for use. The illustration of this process is shown in the composition view of SCS-RA in Figure 4.5.

Figure 4.4 - Logical View of SCS-RA.



SOURCE: Author

Figure 4.5 - Composition View of SCS-RA.



SOURCE: Author

## 4.4 Step RA-4: Architectural Evaluation

The last step to be accomplished for the construction of a RA refers to its evaluation. To further complete this important step, we conducted the evaluation in different ways. First, the reference architecture was verified by checking the requirements with the architectural design. Second, a conceptual validation is presented in which the SCS-RA components were used for mapping the existing SCS. Finally, an validation through the RAModel was performed.

### 4.4.1 Requirements Mapping to the Architectural Design

In Step RA-2 requirements regarding SCS-RA are presented. As a verification these requirements and our architectural design are compared in Table 4.5. For each requirement we describe what part of the reference architecture addresses the requirement. During this mapping, we conclude that all requirements of SCS-RA have correlation with the architectural design of our RA.

Table 4.5 - Requirements Mapping to the Architectural Design

Requirement	Architectural Verification
	The development of SCS through plug-and-play software components is addressed in the SCS-RA part:
TAR-01	<ul style="list-style-type: none"><li>• Definition of components as main structure;</li><li>• Non-dependency of frameworks or application servers to execute the components;</li><li>• Use of components with portability regarding the platform.</li></ul>

Continued on next page

**Table 4.5 – Requirements Mapping to the Architectural Design (cont.).**

Requirement	Architectural Verification
TAR-02	The development of SCS with blocks developed in different programming languages is addressed in the SCS-RA part:
	<ul style="list-style-type: none"><li>• Definition of components as independent of the programming language;</li></ul>
	<ul style="list-style-type: none"><li>• Definition of APIs as a way to provide the components' functionalities;</li><li>• Definition of open-source and standardized protocols as access way to the component.</li></ul>
TAR-03	The development of interoperable SCS is addressed in the SCS-RA part:
	<ul style="list-style-type: none"><li>• Definition of components as independent of the programming language;</li></ul>
	<ul style="list-style-type: none"><li>• Definition of open-source and standardized protocols as access way to the component.</li></ul>
TAR-04	The development of SCS through dynamic architecture is addressed in the SCS-RA part:
	<ul style="list-style-type: none"><li>• Definition of instantiation in the components' level;</li></ul>
	<ul style="list-style-type: none"><li>• Configuration and definition of quality attributes in components' level;</li></ul>
	<ul style="list-style-type: none"><li>• Instantiation of new components considering the computational resources;</li></ul>
	<ul style="list-style-type: none"><li>• Definition of the functionalities in the components' level.</li></ul>

Continued on next page

**Table 4.5 – Requirements Mapping to the Architectural Design (cont.).**

<b>Requirement</b>	<b>Architectural Verification</b>
	<p>The instantiation of SCS through composition of components is addressed in the SCS-RA part:</p> <ul style="list-style-type: none"><li>• Creation of components repository;</li></ul>
TAR-05	<ul style="list-style-type: none"><li>• Creation of Composition component;</li><li>• Creation of Registry component;</li><li>• Registration of components during their execution.</li></ul>
	<p>The development of SCS that allow the use of heterogeneous databases is addressed in the SCS-RA part:</p>
TAR-06	<ul style="list-style-type: none"><li>• Definition of data persistence for each component;</li><li>• Each persistent component is responsible for defining its database.</li></ul>
	<p>The development of SCS that make scalability easy is addressed in the SCS-RA part:</p>
TAR-07	<ul style="list-style-type: none"><li>• Definition of components as independent structures;</li><li>• Scalability in the components' level;</li><li>• Horizontal scalability by functionalities defined in components;</li><li>• Scalability done in different platforms.</li></ul>

Continued on next page

**Table 4.5 – Requirements Mapping to the Architectural Design (cont.).**

Requirement	Architectural Verification
TAR-08	The development of SCS with monitoring and management of quality attributes is addressed in the SCS-RA part:
	<ul style="list-style-type: none"> <li>• Configuration and definition of quality attributes in components' level;</li> </ul>
	<ul style="list-style-type: none"> <li>• Definition of components as main structure;</li> </ul>
	<ul style="list-style-type: none"> <li>• Requirements monitoring and controlling in components' level.</li> </ul>

---

SOURCE: Author.

#### 4.4.2 Validation of the SCS-RA by Mapping of the SCS-RA Components to the Existing Systems

Comparison between the systems functionalities mentioned in Section 3.2 and the architectural design: The experience embodied in successful real-world systems is an important quality parameter to the definition of RA (BAYER et al., 2004). Table 4.6 shows this mapping.

Table 4.6 - Mapping of the SCS-RA components to the Existing Systems

SCS-RA Components	SCOS-2000	SatNet	Hifly	SatCS
Telemetry	✓	✓	✓	✓
Telecommand	✓	–	✓	✓
Tracking	✓	✓	✓	✓
Flight Dynamics	✓	–	✓	–
Rotors	✓	–	✓	–
Orbital Calculator	✓	–	✓	–
Monitoring	✓	✓	✓	✓
Simulation	✓	–	✓	–
Composition	–	–	–	–
Schedule	✓	✓	✓	✓
Mashup	–	–	–	–
Ranging	✓	–	✓	–
Attitude	✓	–	✓	–

Continued on next page

**Table 4.6 – Mapping of the SCS-RA components to the(cont.).**

<b>SCS-RA Components</b>	<b>SCOS-2000</b>	<b>SatNet</b>	<b>Hifly</b>	<b>SatCS</b>
Spacecraft	✓	✓	✓	✓
Ground Station	✓	✓	✓	✓
Administrator	✓	–	✓	✓
Antenna	✓	✓	✓	–
Registry	–	✓	–	–
Discovery	–	–	–	–
Maneuver	✓	–	✓	–
Gateway	–	✓	–	–
Routing	–	✓	–	–

SOURCE: Author.

This comparison showed us that just the Mashup, Discovery, and Composition components are not directly related to the functionalities of these systems. However, they are components that add value to integration and instantiation of systems.

#### 4.4.3 Validation through the RAModel

The RAModel is a reference model specific to RA that provides a complete list of elements needed for constituting a RA. These elements are classified in four groups (NAKAGAWA et al., 2012): Domain, Application, Infrastructure, and Crosscutting. To verify the relevant elements for SCS-RA, we accomplished a comparison with the elements established by the RAModel. Table 4.7 illustrates part of such comparison.

Table 4.7 - Comparison between the elements of RAModel and SCS-RA.

<b>Elements</b>	<b>Description</b>	<b>SCS-RA</b>
<b>Application Group</b>		
Functional requirements	Set of functional requirements that are common in systems developed using this architecture.	✓
Goal and needs	Intention of the reference architecture and needs that could be covered by the reference architecture.	✓
Constraints	Constraints presented by the reference architecture and/or constraints in specific part of a reference architecture.	✓

Continued on next page

**Table 4.7 – Comparison between the elements (cont.).**

<b>Elements</b>	<b>Description</b>	<b>SCS-RA</b>
Domain Data	Common data found in systems of the domain. These data are presented in a higher level of abstraction, considering the higher level of abstraction of the reference architecture.	✓
Limitations	Limitations presented by the reference architecture and/or limitations in specific part of a reference architecture.	✓
Risks	Risks in using the reference architecture and/or risks in using some part of such architecture.	- -
Scope	Scope that is covered by the reference architecture, i.e., the set of systems developed based on the reference architecture.	✓
<b>Domain Group</b>		
Legislations, standards and regulations	Laws, standards, and regulations existing in the domain that should be present in systems resulted from the reference architecture.	- -
Quality attributes	Quality attributes, for instance, maintainability, portability, and scalability, that are desired in systems resulted from the reference architecture.	✓
System compliance	Means to verify if systems developed from the reference architecture follow existing legislations, standards, and regulations.	- -
<b>Infrastructure Group</b>		
General structure	General structure of the reference architecture, represented sometimes by using existing architectural styles.	✓
Software elements	Elements of software present in the reference architecture, e.g., subsystems and classes, which could be used to develop software systems.	✓
Best practices and guidelines	Well-experimented practices to develop systems of the domain, These practices could be accompanied by guidelines describing how to apply these practices.	✓

Continued on next page



**Table 4.7 – Comparison between the elements (cont.).**

<b>Elements</b>		<b>Description</b>	<b>SCS-RA</b>
Hardware elements	ele-	Elements of hardware, such as server and devices, which host systems resulted from the reference architecture.	- -
<b>Crosscutting Group</b>			
General structure	struc-	General structure of the reference architecture, represented sometimes by using existing architectural styles.	✓
Internal communication		Means by which occur exchange of information among internal parts of systems resulted from the reference architecture.	✓
Decisions		Decisions, including description of the decision, options (alternatives), rationale, and tradeoffs, must be reported during the development of the reference architecture.	- -
Domain terminology	Termi-	Set of terms of the domain that are widely accepted by the community related to that domain and are, therefore, used in the description of the reference architecture.	- -

SOURCE: Author.

As a result, we can notice that SCS-RA conforms to almost all elements of RAModel. Nevertheless, some of the missing elements refer to limitations of SCS-RA and other factors such as risks, decisions, terminology, legislations and system compliance, which are due to the particularities of the application domain and can still be further added to this RA.

## 4.5 Final Remarks

This chapter presented SCS-RA, a reference architecture for SCS. The establishment of SCS-RA was guided by ProSA-RA. Several sources of information were investigated and requirements were defined according to two groups: Architectural Domain Requirements and Technical Architectural Requirements. SCS-RA was described in views, represented in semi-formal notations by ArchiMate and UML language, to address different concerns and stakeholders. The following views were described: enterprise, structural, data flow, logical, and composition.

SCS-Ra has the capacity to compose systems in which: components may be written in different programming languages, components may be running concurrently in a distributed, heterogeneous environment without shared address spaces, and architectures may be changed at runtime. SCS-RA contributes to support the development and evolution of SCS, contributing to improve interoperability, structuring and maintaining SCS, and contributing to improve reuse of software components when developed based on SCS-RA.

The evaluation of SCS-RA was accomplished through the requirements mapping to the architectural design, mapping of the SCS-RA components to the existing systems, and through the use RAModel. In addition to, the architectural evaluation, we also conducted a case study to provide evidences on the viability of SCS-RA and observe possible benefits of its adoption. This case study is discussed in the next chapter.

## 5 CASE STUDY - USING SCS-RA FOR DEVELOPMENT OF MICROSATELLITES CONTROL SYSTEM

In this chapter we present a case study of the use of SCS-RA in the development of a SCS. We instantiated a concrete software architecture in conformance with our RA, and we proceeded subsequently with the development of a software product for a MicroSatellites Control System (MicroSatCS). The development of MicroSatCS contributed to observe benefits of adopting SCS-RA and to supply evidence on its viability.

We instantiated a concrete software architecture in conformance with our RA, and we proceeded subsequently with the software development of a software product for a MicroSatellites Control System. The MicroSatCS Development Process was based on the life cycle defined by ECSS in [EUROPEAN COOPERATION FOR SPACE STANDARDIZATION \(ECSS\)](#) (2004). Section 5.2 shows an overview of The MicroSatCS Development Process. Section 5.3 briefly presents the Architectural Design Process. Section 5.4 briefly presents the Software Design & Implementation Process. Section 5.5 shows the Software Operation Process.

### 5.1 MicroSatCS Overview

Currently, as a consequence of considerable amount of projects and prospects of the increasing number of microsatellites, a new control system for this particular type of satellite needs to be developed. Thus, some characteristics were considered during the development: (i) a large amount of small satellites imposes a greater computational and operational capacity to the MicroSatCS; (ii) the operations in shared ground stations increase the coordination burden for monitoring and control of the satellites; (iii) the lifetime of small satellites is usually smaller when compared to the larger ones; and (iv) the development of MicroSatCS must be compatible with the limited budget for small satellites missions.

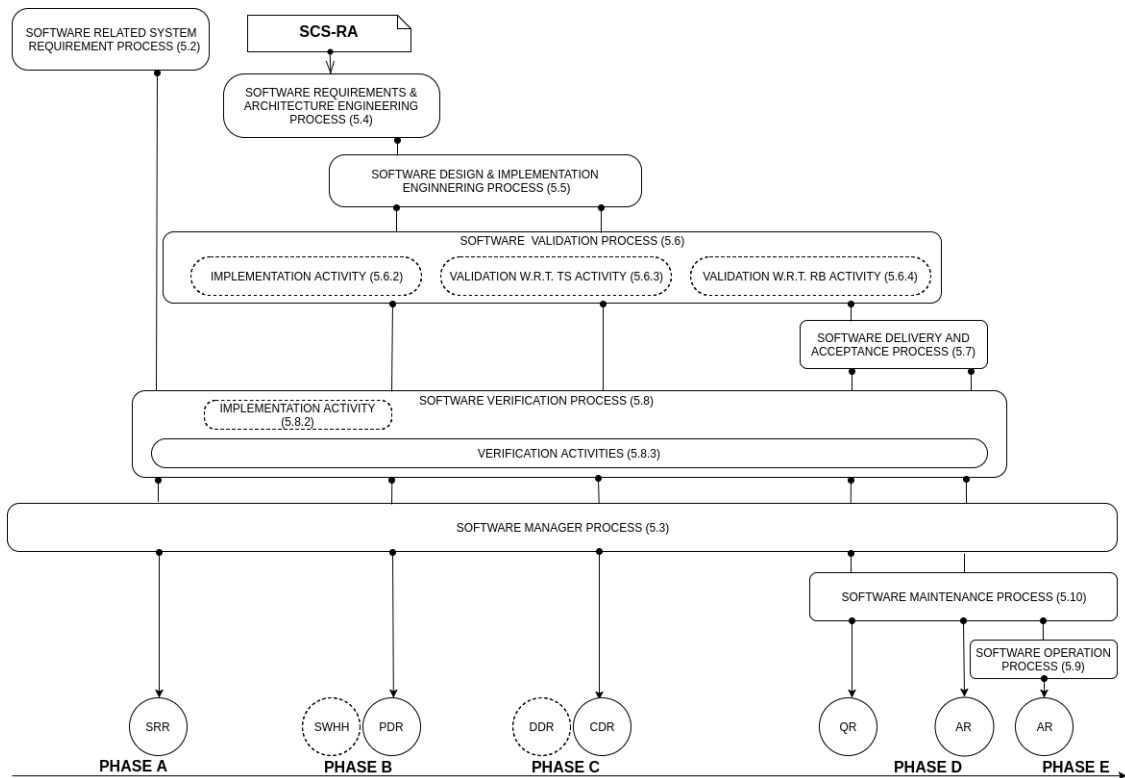
### 5.2 The MicroSatCS Development Process

In systems engineering, the conceptual design of a space system is constituted into distinct phases, as presented in ([NASA, 2007](#); [EUROPEAN COOPERATION FOR SPACE STANDARDIZATION \(ECSS\), 2004](#)). Each phase is dominated by a main activity and finishes with a review of the work done. The software development process for SCS, when included in this context, should correlate to these phases.

The development of MicroSatCS was based on the life cycle defined by ECSS in [EU-](#)

ROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS) (2004), which consists of phases: 0: Mission analysis/needs identification; A: Feasibility; B: Preliminary definition; C: Detailed definition; D: Qualification and production; E: Operations/utilization; and F: Disposal. The transition of one phase to the next one is controlled by reviews (System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), Qualification Review (QR), Acceptance Review (AR), and Operational Readiness Review (ORR)). These reviews are relevant to the software engineering standard process defined by ECSS in ECSS (2009b), which offers the possibility to anticipate the PDR in a Software Requirements Review (SWRR) and the CDR into a Design Dedicated Reviews (DDR). In compliance with these standards, Figure 5.1 shows the MicroSatCS development life cycle. The architectural design process was guided by SCS-RA, as detailed in the sequence.

Figure 5.1 - MicroSatCS Development Life Cycle.

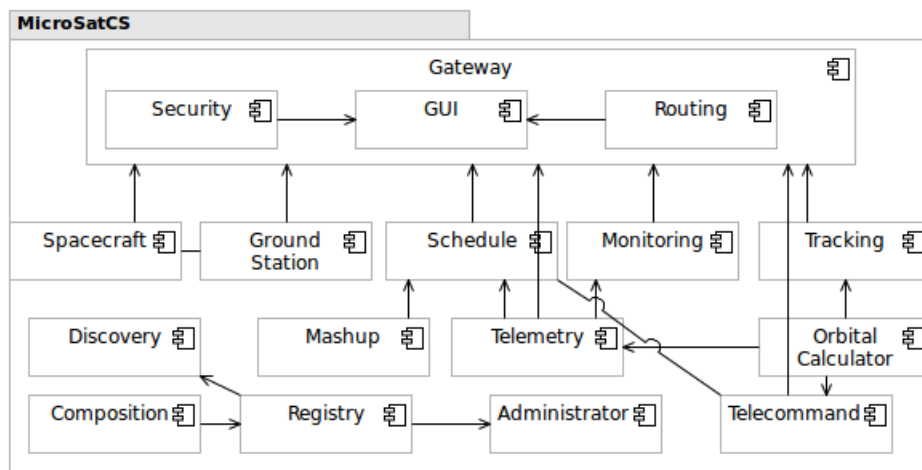


SOURCE: Adapted from ECSS (2009b).

### 5.3 Architectural Design Process

The main activity carried out during this process involved identifying and representing major system components and their communications. The components necessary to meet the functional and non-functional requirements of MicroSatCS were identified from SCS-RA, which defines all components to the satellites control domain. The components that make up MicroSatCS are shown in Figure 5.2 through of a logical view.

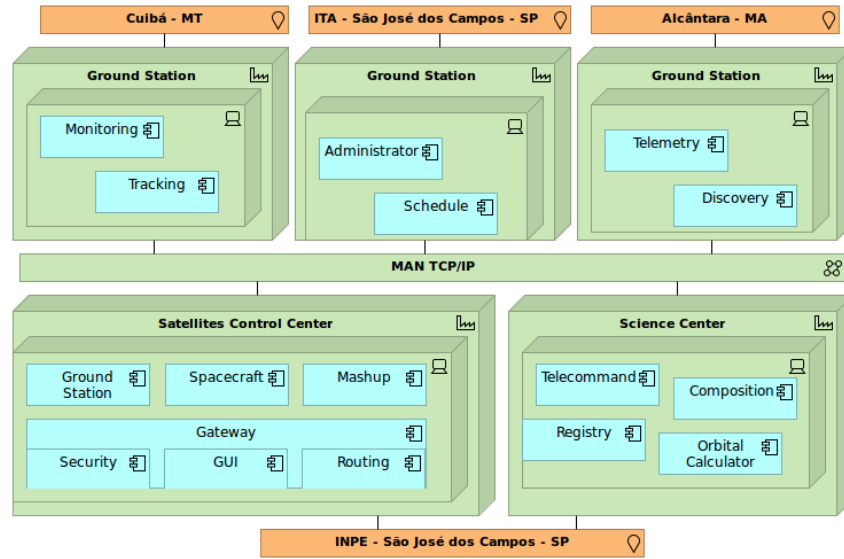
Figure 5.2 - Logic View of MicroSatCS.



SOURCE: Author

MicroSatCS is a distributed system, its components have the capability of dynamically changing their place of execution and to scale according to the performance requirements and the capability of servers resources. This contributes to improve interoperability, scalability, and sharing of terrestrial resources. Figure 5.3 shows a deployment view considering our organizational structure, which has three ground stations located in the cities of Cuibá in the state of Mato Grosso do Sul, Alcântara in the state of Maranhão, and São José dos Campos in the state of São Paulo. Besides that, the Science Center and SCC also in São José dos Campos.

Figure 5.3 - Deployment View of MicroSatCS.



SOURCE: Author

After the architectural design process has been completed, the difference between estimated effort (which did not consider the use of SCS-RA) and real effort was analyzed. This analysis considered mainly quantitative data and results achieved showed a reduction of 65% in terms of person-hours effort. This reduction occurred mainly during the definition of architectural requirements, in the definition of styles and patterns, and in the production of architectural models.

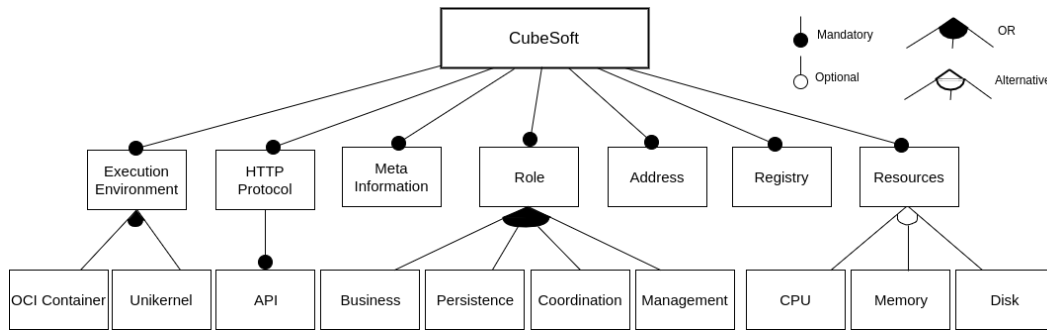
#### 5.4 Software Design & Implementation Process

This process consisted of designing software items, coding them, and also doing integration. During the detailed design, an analysis was performed to define the software component model. Component models have been introduced in different domains (LAU; WANG, 2007; CRNKOVIC et al., 2011; LAU; WANG, 2005). For the domain of space systems, some initiatives have been presented for satellite on-board software (PANUNZIO, 2011; PANUNZIO; VARDANEGA, 2010). Domain components are the most difficult kind of component for developing and are more costly because they need a huge infrastructure for being deployed, but, productivity may increase 1000% (HEINEMAN; COUNCILL, 2001).

The definition of the domain component for SCS considered the technical require-

ments of interoperability, independence, and scalability described in SCS-RA. Thus, to meet these requirements in a satisfactory degree of completeness was defined a Software Component Model for Satellites Control Domain (CubeSoft). CubeSoft component model is an abstractly described component model, its definition is intended to guide the component implementation process for SCS. CubeSoft is non-proprietary, language and platform independent component architecture. It allows application programmers to design and implement a distributed application independent of specific programming languages, operating systems or vendor-specific communication infrastructures. Figure 5.4 shows a feature diagram with the CubeSoft taxonomy.

Figure 5.4 - CubeSoft Taxonomy



SOURCE: Author

In this diagram, the features and subfeatures of CubeSoft are depicted. A detailed description was accomplished in a component specification document. CubeSoft allows the implementation of components in different programming languages, execution in separate and isolated process address spaces, scalability at component level, portability of the execution platform through its execution environment. Moreover, we defined additional architectural decisions to meet these requirements, as follows:

- a) The environment for deployment and execution is defined through the use of Open Container Initiative (OCI)<sup>1</sup> or Unikernel<sup>2</sup>. CubeSofts for MicroSatCS used the specifications OCI as execution environment, which has portability in kernel level and favors the execution of components in inde-

<sup>1</sup><https://www.opencontainers.org>

<sup>2</sup><http://unikernel.org/>

pendent platforms. Quality attributes such as, isolation and security, are the responsibility of this environment. Scalability was managed by Swarm Cluster<sup>3</sup>;

- b) Applications programming interfaces (APIs) are defined by Interface Description Language (IDL) for RESTful and gRPC<sup>4</sup>. Access is via HTTP protocol and the documentation was made using OPENAPI<sup>5</sup> specification. These standards are open and widely adopted, what can contribute to improve interoperability;
- c) Meta Information about the component must be made available during the component deployment and execution process. Examples are: execution IP address, computational resources, and configurations for deployment.
- d) The application of SCS-RA enables developing components in different technologies or reusing components. CubeSofts developed are: (i) Orbital Calculator using Python<sup>6</sup> and Flask<sup>7</sup>; (ii) Telemetry, Telecommand, Tracking, Flight Dynamic, Routing, Schedule, Ground Station, Composition, Security, Mashup using Java<sup>8</sup>, Spring Framework<sup>9</sup>; and (iii) GUI using JavaScript, NodeJs<sup>10</sup>, CSS, HTML, AngularJs<sup>11</sup>; Components reused are: (i) Discovery and Registry from Netflix/eureka<sup>12</sup>; (ii) Administrator from Rancher<sup>13</sup>; and (iii) Monitoring from Open MCT framework <sup>14</sup> (an open source framework developed at NASA's Ames Research Center);
- e) CubeSoft must have an access address to make your available interfaces available and their functionality . Access must be made through IP addressing and an access port. The IP address can be local or external for access between hosts;
- f) CubeSoft must provide mechanisms to perform the registration during its execution. This registry contributes to identify your location and to manage your computing resources.

---

<sup>3</sup><https://docs.docker.com/engine/swarm>

<sup>4</sup><https://grpc.io>

<sup>5</sup><https://www.openapis.org>

<sup>6</sup><https://www.python.org/>

<sup>7</sup><http://flask.pocoo.org>

<sup>8</sup><http://www.java.com>

<sup>9</sup><https://spring.io>

<sup>10</sup><https://nodejs.org>

<sup>11</sup><https://angularjs.org>

<sup>12</sup><http://github.com/Netflix/eureka>

<sup>13</sup><http://rancher.com>

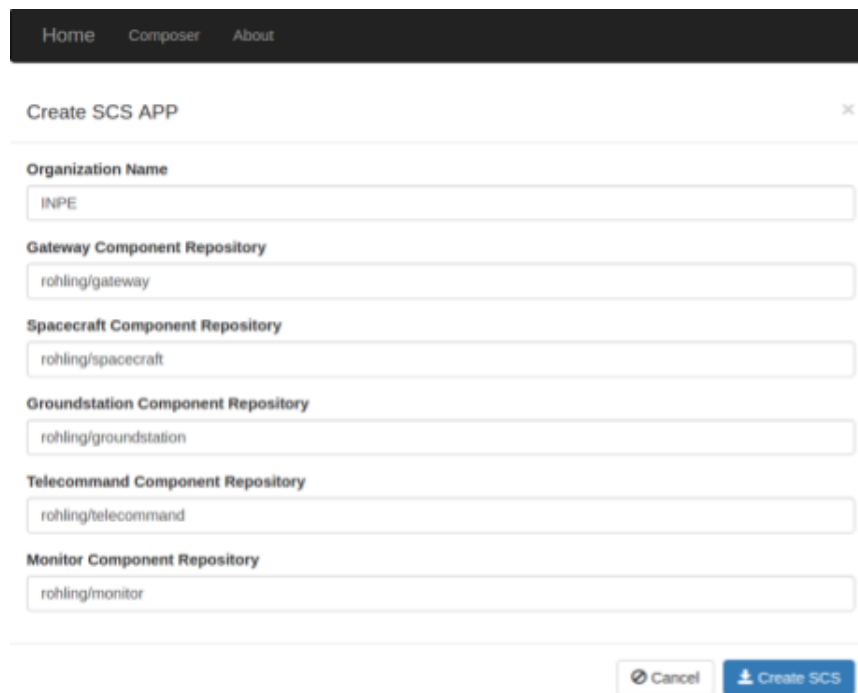
<sup>14</sup><https://nasa.github.io/openmct>



## 5.5 Software Operation Process

This section briefly presents the MicroSatCS in operation. MicroSatCS was instantiated visually from the Composition component, as shown in Figure 5.5. The instantiation allows to define computational resources such as memory, CPU and disk for each component. The components were stored in the Docker repository<sup>15</sup>.

Figure 5.5 - Instantiation of MicroSatCS.



The screenshot shows a web interface for creating a SCS application. At the top, there is a navigation bar with links for 'Home', 'Composer', and 'About'. Below this, the main heading is 'Create SCS APP' with a close button (X) on the right. The form contains several input fields, each with a label and a text box: 'Organization Name' (containing 'INPE'), 'Gateway Component Repository' (containing 'rohling/gateway'), 'Spacecraft Component Repository' (containing 'rohling/spacecraft'), 'Groundstation Component Repository' (containing 'rohling/groundstation'), 'Telecommand Component Repository' (containing 'rohling/telecommand'), and 'Monitor Component Repository' (containing 'rohling/monitor'). At the bottom right of the form, there are two buttons: 'Cancel' and 'Create SCS'.

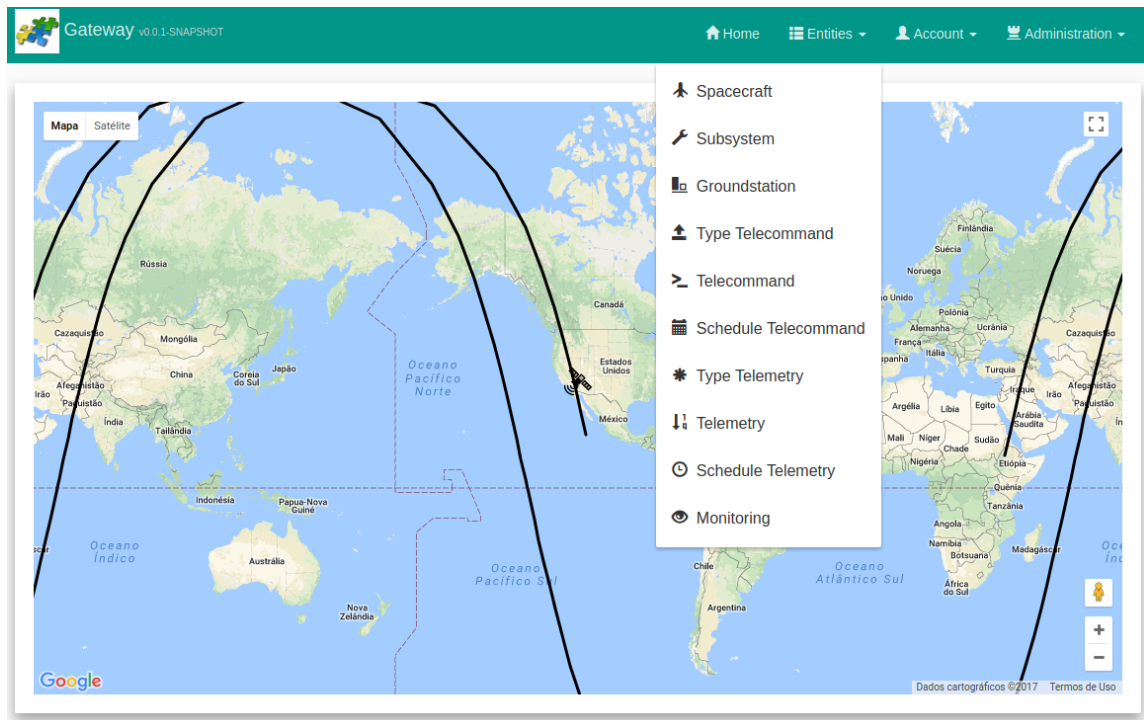
SOURCE: Author

The GUI component provides the operations of MicroSatCS through a browser and Figure 5.6 shows some typical operations of MicroSatCS and, in particular, the tracking of UbatubaSat Picosatellite.

The computational resources consumed by the components during their execution, such as capacities of memory, CPU, network, and storage, are assigned according to each components requirements. The Administrator component monitors and manages the computational resources of each component as Figure 5.7 shows, for example, those being the resources consumed by Gateway Component in operation.

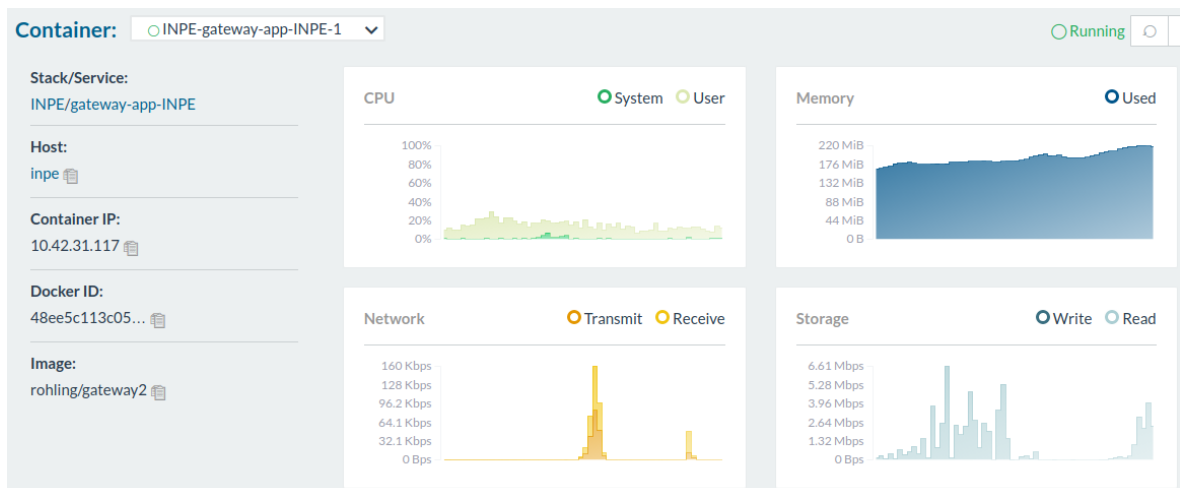
<sup>15</sup><https://hub.docker.com>

Figure 5.6 - MicroSatCS and Tracking of Satellite.



SOURCE: Author

Figure 5.7 - Computational Resources - Gateway Component

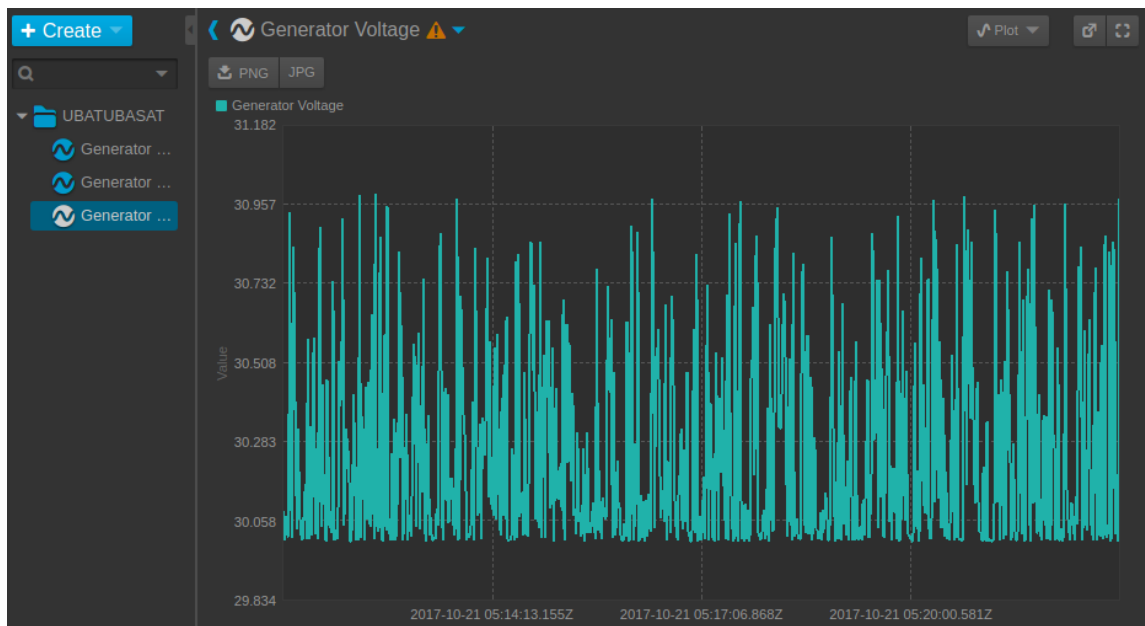


SOURCE: Author

In the operation of satellites, it is very important to monitor the health status of the systems for detecting any anomalies in the housekeeping data as soon as possible. Monitoring Component can provide data visualization about such health status in different ways.

The application of SCS-RA and use of CubeSoft enable developing components in different technologies or reusing components. Thus, the Monitoring component was developed from Open MCT framework<sup>16</sup>, an open source framework for mission control. It is developed at NASA's Ames Research Center in collaboration with the Jet Propulsion Laboratory. Figure 5.8 shows the telemetry data of voltage of Tancredo-I Picosatellite.

Figure 5.8 - Monitoring Component - Tancredo-I picosat telemetry in the UbatubaSat Project.



SOURCE: Author

## 5.6 Final Remarks

In this section briefly presents the application of SCS-RA, showing its usefulness and benefits, during the development of MicroSatCS, in particular during the de-

<sup>16</sup><https://nasa.github.io/openmct>

sign, implementation and operation processes. Results indicate that SCS-RA can contribute to the development of SCS software architectures. However, other case studies are necessary and will be conducted to confirm the obtained results, identify possible improvements for the reference architecture, and increase the confidence on its adoption.

The next chapter concludes this thesis, summarizing the main contributions, discussing general limitations, and mentioning future work.

## 6 CONCLUSIONS AND FUTURE WORKS

### 6.1 Discussion

The establishment of SCS-RA requires a deep knowledge on diverse subjects from different stakeholders, as well as, the need of an initial investment with regard to time and cost. However, a significant reduction of time and effort was observed during the comparison of the initial effort estimate, which did not consider the use of SCS-RA, with the real effort of development with the use of SCS-RA. This analysis considered mainly quantitative data and results achieved showed an effort reduction of 65% in terms of person-hours during architectural design process. In addition to the effort reduction observed in our analysis, a cost-benefit analysis could be carried out through a particular economic model for RA ([MARTÍNEZ-FERNÁNDEZ et al., 2013](#)).

Using SCS-RA, best practices of SCS development increase the productivity of developers. MicroSatCS functionalities were more easily built with components developed and provided from different providers. Hence, MicroSatCS has benefited from reduced development and integration effort, besides reuse, which is an approach that has several other benefits ([LAU; WANG, 2007](#); [LI et al., 2007](#); [MILI et al., 1995](#)).

### 6.2 Final Remarks

In this thesis, we presented a RA for SCS, referred to as SCS-RA. The main contribution of SCS-RA lies in providing guidelines for the development and evolution of SCS. Therefore, these systems could be easily built from this architecture, reducing time, efforts, and rework and improving interoperability and sharing of ground resources among space organizations. SCS-RA was proposed following a systematic process and during the evaluations performed, it was observed that SCS-RA is complete, since it presents the most important elements that should be present in a RA. In addition, the development of MicroSatCS contributed to observe the benefits of adopting SCS-RA and to supply evidence on its viability.

An important contribution of this work is to make the reference architecture available for any government or research institution that intends to create a satellite architecture control system. We deliver knowledge and expertise about this domain encapsulated as a reference architecture, and make it available for world-wide adoption and improvement, fostering the advancements of sciences and supporting the development of space technologies for other countries.

Results achieved from usage of SCS-RA during the development of a Microsatellite Control System for INPE, a Brazilian National Institute for Space Research, showed a significant reduction of time and effort. Furthermore during operational phase, benefits of interoperability, scalability, and sharing of ground resources were observed. As a RA gathers knowledge acquired from a domain and structures it in a reusable and extensible form (GRACIANO NETO et al., 2015), forthcoming software products for SCS could be inherently interoperable, as they could be instantiated from a common RA, following shared standards, and facilitating resources sharing. As a consequence, several other SCS could be built from SCS-RA.

Besides the positive results obtained so far, we highlight the need to conduct a more complete evaluation involving different integrated SCS. From this work, it can be concluded that there are still challenges to be met, especially, in the description, maintenance, and sustainability of SCS-RA to be performed by several space organizations.

### 6.3 List of Publications Attained

Excerpts of this thesis have been either published or submitted for the appreciation of editorial boards of journals, conferences, symposiums, and workshops, according to the abstracts and articles presented below.

**Event:** II Latin American IAA CubeSat Workshop. February 28 to March 2, 2016. Ingleses Beach, Florianopolis, Brazil

**Title:** Integration of the INPE Ground Station into the SatNet Network

**Authors:** Ricardo Tubío-Pardavia, Jorge Enrique Espindola Diaz, Adair José Rohling, Mauricio Gonçalves Vieira Ferreira, Walter Abrahao Dos Santos, Jordi Puig-Suari, Fernando Aguado-Agelet.

**Abstract:** The researchers of the Ground Station at the National Institute for Space Research (INPE) in Brazil have started working, together with the main developers of the SATNet network, in the integration of their ground segment into that very same network. This paper presents the description of that integration process together with the results obtained and with a description of the current and upcoming small satellite projects in Brazil and in Latin America. The SATNet network aims at incorporating the capabilities of all the already deployed university Ground Stations into a single, coherent and usable resource. The approach used for this network is based on heterogeneity, allowing the integration of very different

ground stations into a single ground system.

**Event:** 7º Workshop em Engenharia e Tecnologia Espaciais 2016, September 23-24, 2016. São Jose dos Campos, Brazil.

**Title:** CubeSoft : Componentes de Software para Desenvolvimento de Sistemas de Controle de Satélites

**Authors:** Adair José Rohling, Mauricio Gonçalves Vieira Ferreira, Walter Abrahao Dos Santos.

**Abstract:** Aplicações do domínio de controle de satélites são complexas e estão em constante evolução em consequência de avanços tecnológicos. Atender requisitos de portabilidade, desenvolvimento e reúso de componentes torna-se um desafio, principalmente para aplicações monolíticas que utilizam camadas como estrutura principal de decomposição e para aplicações desenvolvidas em diferentes modelos de componentes de software. Neste contexto, este trabalho propõe o uso de CubeSofts como estrutura principal para aplicações deste domínio. CubeSofts são componentes de software definidos através de unidades funcionais independentes encapsuladas em containers. O estilo arquitetural microservices é aplicado na definição de CubeSofts como estrutura principal das aplicações e containers são usados como ambiente de execução para diferentes modelos de componentes. Uma arquitetura de software é apresentada como resultado da aplicação de CubeSofts em sistemas de controle de satélites.

**Event:** 26th Annual INCOSE International Symposium (IS2016) Edinburgh, July 18-21, 2016

**Title:** Concurrent Structured Analysis SE method applied to a solar irradiance monitor satellite

**Authors:** Rodrigo Britto Maria, Halph Macedo Fraulob, Adrielle Chiaki, Gabriel Gustavo Coronel Mariño, Adair José Rohling, Geilson Loureiro.

**Abstract:** The traditional Systems Engineering approach on systems development leans toward the product in development, in all levels of abstraction, while the traditional concurrent engineering approach focuses on both product and organization simultaneous development, although considering only lower levels of abstraction. For complex systems, the Concurrent Structured Analysis method is a better fit for the SE process, since it considers both the product and organization simultaneous de-

velopment, in all levels of abstraction. The underlying concepts of this method were applied to the development of a solar irradiance monitor satellite. Several analyses were performed on selected scenarios of the system life cycle processes to exemplify the application of the concepts. The final outcome was a proposal design of the product and the organization needed to implement its life cycle processes. It was noted that the same scenario can influence the product and the organization, both requiring to be considered in the analysis.

**Event:** Symposium on Applied Computing, Pau, France, April 9–13, 2018 (SAC’18).

**Title:** Externalizing Patterns for Simulations in Software Engineering of Systems-of-Systems

**Authors:** Valdemar Vicente Graciano Neto, Wallace Manzano, Adair José Rohling, Mauricio Gonçalves Vieira Ferreira, Tiago Volpato, Elisa Yumi Nakagawa.

**Abstract:** Systems-of-Systems (SoS) often support critical domains. They must be trustworthy, i.e., they must keep their operation in progress, being not subject to failures, as they can cause potential damages and hazards to human integrity. Simulations are a recurrent approach in SoS development, as they can anticipate potential failures, consequently increasing the level of trustworthiness and quality exhibited by a SoS. Nevertheless, simulation is still software and demands engineering. Moreover, many simulation formalisms are not trivial of specifying, sometimes tangling software and hardware details to program an executable simulation. Thus, the aim of this paper is contributing for software engineering of SoS by externalizing two patterns for the conception of SoS simulations. We evaluated our patterns by applying them in a case study in two different domains. For both, patterns were successfully applied during automatic generation of functional code, supporting the execution of SoS simulations and prediction of SoS behavior at design-time.

**Event:** XIV Simpósio Brasileiro de Sistemas de Informação (SBSI 2018).

**Title:** A Study on Goals Specification for Systems-of-Information Systems: Design Principles and Conceptual Model

**Authors:** Valdemar Vicente Graciano Neto, Flavio Horita, Everton Cavalcante, Adair José Rohling, Jamal El-Hachem, Daniel Santos, Elisa Yumi Nakagawa.

**Abstract:** Software-intensive information systems can be aggregated to form *Systems-of-Information Systems* (SoIS) and provide novel functionalities to achieve



high-level goals, also known as missions. Missions represent an important concern in this context since they are related to both capabilities of constituent systems and how they shall interact with each other within a SoIS. Due to such a relevant role, we conducted an exploratory study to evaluate how a state-of-the-art language for mission specification has supported mission (renamed as *goal* in this study) specification for SoIS. This investigation has been carried out in the context of a space SoIS composed of independent ground information systems, satellites and other systems to provide territory monitoring and environmental data distribution. Results indicate a lack of support for the specificities of SoIS goals, such as the representation of interdependency among activities and dynamicity support. As main contributions, this paper comes up with a set of design principles and a corresponding conceptual model to be followed by languages tailored to support goal specification in SoIS.

**Journal:** Innovations in Systems and Software Engineering - A NASA Journal

**Title:** A Reference Architecture for Satellite Control Systems

**Authors:** Adair José Rohling, Valdemar Vicente Graciano Neto, Mauricio Gonçalves Vieira Ferreira, Walter Abrahão Dos Santos, Elisa Yumi Nakagawa.

**Abstract:** Software for Satellite Control Systems (SCS) domain performs a relevant role in space systems, being responsible for ensuring the functioning of the satellites, from the orbit launch to the end of their lifetime. Systems in this domain are complex and are constantly evolving due to technological advancement of satellites, the significant increase of controlled satellites, and the interoperability among space organizations. However, in order to meet such complexity and such evolution, the architectures of these systems have been usually designed in an isolated way by each organization, hence may be prone to recurrent efforts and difficulties of interoperability. In parallel to this scenario, reference architecture, a special type of software architecture that aggregates knowledge of a specific domain, has performed an important role for the success in development, standardization, and evolution of systems in several domains. Nevertheless, the usage of reference architecture has not been explored in the SCS domain. Thus, this article presents a Reference Architecture for Satellite Control Systems (SCS-RA). Results achieved from usage of SCS-RA in the development of a Microsatellite Control System for National Institute for Space Research (INPE) showed a significant reduction of effort, benefits of interoperability, scalability, and sharing of ground resources.



## REFERENCES

ALMEIDA, E. S. de. **RiDE: The RiSE process for domain engineering**. PhD Thesis (Ph.D. Thesis) — Federal University of Pernambuco, Recife-PE, Brazil, 2007. 39

AMERICA, P.; OBBINK, H.; OMMERING, R. van; LINDEN, F. van der. Copam: A component-oriented platform architecting method family for product family engineering. In: \_\_\_\_\_. **Software product lines: experience and research directions**. Boston, MA: Springer US, 2000. p. 167–180. ISBN 978-1-4615-4339-8. Available from: <[https://doi.org/10.1007/978-1-4615-4339-8\\_9](https://doi.org/10.1007/978-1-4615-4339-8_9)>. 39

ANGELOV, S.; GREFEN, P.; GREEFHORST, D. A classification of software reference architectures: analyzing their success and effectiveness. In: JOINT WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE, 2009. **Proceedings...** Cambridge: IEEE, 2009. p. 141–150. 27

\_\_\_\_\_. A framework for analysis and design of software reference architectures. **Information and Software Technology**, v. 54, n. 4, p. 417–431, apr. 2012. ISSN 0950-5849. Available from: <<http://dx.doi.org/10.1016/j.infsof.2011.11.009>>. 29, 31

ANGELOV, S.; TRIENEKENS, J. J.; GREFEN, P. Towards a method for the evaluation of reference architectures: experiences from a case. In: EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE (ECSA), 2., 2008. **Proceedings...** Berlin: Springer, 2008. p. 225–240. ISBN 978-3-540-88029-5. Available from: <[http://dx.doi.org/10.1007/978-3-540-88030-1\\_17](http://dx.doi.org/10.1007/978-3-540-88030-1_17)>. 31

ARZA, M.; DREIHAHN, H. SLE Routing – simplified station access for mission operations. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 2012. **Proceedings...** Stockholm, 2012. Available from: <<http://www.spaceops2012.org/proceedings/documents/id1294305-Paper-001.pdf>>. 2

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Campo CCSDS para identificação global de espaçonaves: procedimentos de controle para atribuição de códigos - CCSDS 320.0-B-5-S**. São José dos Campos, 2007. 29 p. Access in: 19 dez. 2017. 70

ATKINSON, C.; BAYER, J.; BUNSE, C.; KAMSTIES, E.; LAITENBERGER, O.; LAQUA, R.; MUTHIG, D.; PAECH, B.; WÜST, J.; ZETTEL, J.

**Component-based product line engineering with UML.** Boston, MA, USA: Addison-Wesley Longman Publishing, 2002. ISBN 0-201-73791-4. 40

ATKINSON, C.; BAYER, J.; MUTHIG, D. Component-based product line development: the kobra approach. In: PROCEEDINGS OF THE FIRST CONFERENCE ON SOFTWARE PRODUCT LINES : EXPERIENCE AND RESEARCH DIRECTIONS: EXPERIENCE AND RESEARCH DIRECTIONS, 1., 2000. **Proceedings...** Norwell, MA: Kluwer Academic Publishers, 2000. p. 289–309. ISBN 0-79237-940-3. Available from:

<<http://dl.acm.org/citation.cfm?id=355461.357556>>. 40

BACHMANN, F.; BASS, L.; BUHMAN, C.; COMELLA-DORDA, S.; LONG, F.; ROBERT, J.; SEACORD, R.; WALLNAU, K. **Technical concepts of component-based software engineering.** 2. ed. Pittsburgh, PA, 2000.

Available from:

<<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5203>>.

38

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice.** 2. ed. Boston, MA, USA: Addison-Wesley Longman Publishing, 2003. ISBN 0321154959. 17, 28, 47

BAYER, J.; GANESAN, D.; GIRARD, J.-F.; KNODEL, J.; KOLB, R.; SCHMID, K. **Definition of reference architecture based on existing systems.** [S.l.], 2004. Available from:

<<http://publica.fraunhofer.de/dokumente/N-21572.html>>. 31, 80

BOLEA-ALAMANAC, A. A. . L. A. . A. **ISIS: ISU small satellite interdisciplinary survey.** 2001. Available from: <[https://isulibrary.isunet.edu/opac/index.php?lvl=notice\\_display&id=4686](https://isulibrary.isunet.edu/opac/index.php?lvl=notice_display&id=4686)>. 9, 11, 12

BORN, M.; HOFFMANN, A.; RENNOCH, A.; REZNIK, J.; RITTER, T.; VOUFFO, A. The european corba components open source initiative. **ERCIM News n.55**, Oct 2003. Available from:

<[https://www.ercim.eu/publication/Ercim\\_News/enw55/rennoch.html](https://www.ercim.eu/publication/Ercim_News/enw55/rennoch.html)>. 37

BOUWMEESTER, J.; GUO, J. Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology. **Acta Astronautica**, v. 67,

n. 7, p. 854 – 862, 2010. ISSN 0094-5765. Available from: <<http://www.sciencedirect.com/science/article/pii/S0094576510001955>>. 11

BRITO, P. H. da S.; GUERRA, P. A. de C.; RUBIRA, C. M. F. **Estudo sobre estilos arquiteturais para sistemas de software baseados em componentes**. [S.l.], March 2007. 17

BROWN, A. W.; SHORT, K. On components and objects: the foundations of component-based development. In: INTERNATIONAL SYMPOSIUM ON ASSESSMENT OF SOFTWARE TOOLS AND TECHNOLOGIES, 5., 1997. **Proceedings...** [S.l.], 1997. p. 112–121. 33, 38, 39

BROY, M.; DEIMEL, A.; HENN, J.; KOSKIMIES, K.; PLASIL, F.; POMBERGER, G.; PREE, W.; STAL, M.; SZYPERSKI, C. A. What characterizes a (software) component? **Software - Concepts and Tools**, v. 19, n. 1, p. 49–56, 1998. 33

CALPOLY. **CubeSat design specification rev. 13**. California, 2014. Available from: <[http://www.cubesat.org/s/cds\\_rev13\\_final2.pdf](http://www.cubesat.org/s/cds_rev13_final2.pdf)>. Access in: 05 Feb. 2016. 9, 10

CAMPBELL, A. T.; CHOU, S. T.; KOUNAVIS, M. E.; STACHTOS, V. D.; VICENTE, J. Netbind: a binding tool for constructing data paths in network processor-based routers. In: IEEE OPEN ARCHITECTURES AND NETWORK PROGRAMMING PROCEEDINGS (OPENARCH), 2002. **Proceedings...** [S.l.], 2002. p. 91–103. 35

CASTRO, A.; PAGE, H.; WALKER, R.; EMMA, F.; AGUADO, F.; VAZQUEZ, A. J. Connecting students with space: genso pre-operational activities and preparation for geoid/humsat operations. **ESA bulletin**, v. 2012, p. 38–43, 02 2012. 56

CHAMOUN, J. P.; RISNER, S.; BEECH, T.; GARCIA, G. Bridging ESA and NASA worlds: lessons learned from the integration of Hifly ®/SCOS-2000 in NASA's GMSEC. In: IEEE AEROSPACE CONFERENCE, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 1–8. ISSN 1095-323X. 3, 52, 55

CHEESMAN, J.; DANIELS, J. **UML components - a simple process for specifying component-based software**. [S.l.]: Addison-Wesley, 2001. (Component Software Series). 39

CHIN, A.; COELHO, R.; NUGENT, R.; MUNAKATA, R.; PUIG-SUARI, J. CubeSat: the pico-satellite standard for research and education. In: AIAA SPACE CONFERENCE AND EXPOSITION, 2008. **Proceedings...** AIAA, 2008.

Available from: <<https://doi.org/10.2514/6.2008-7734>>. 10

CHIN, E.; BROOKS, L.; NUGENT, R.; PUIG-SUARI, D. J.; POLY, C.; OBISPO, S. L.; CHIN, A.; COELHO, R.; BROOKS, L.; NUGENT, R.; JORDI, D.; SUARI, P. Standardization promotes flexibility: a review of cubesats. In: AIAA RESPONSIVE SPACE CONFERENCE, 6., 2008. **Proceedings...** [S.l.]: AIAA, 2008. p. 1–6. 11

CLEMENT, S. J.; MCKEE, D. W.; XU, J. Service-oriented reference architecture for smart cities. In: IEEE SYMPOSIUM ON SERVICE-ORIENTED SYSTEM ENGINEERING, 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 81–85. 3, 29

CLEMENTS, P. **Documenting Software Architectures: Views and Beyond**. Addison-Wesley, 2003. (SEI series in software engineering). ISBN 9780201703726. Available from: <<https://books.google.com.br/books?id=ASc9HYPkr4sC>>. 24

CLEMENTS, P.; COHEN, S.; DONOHOE, P.; NORTHROP, L. **Control channel toolkit: a software product line case study**. Pittsburgh, PA, 2001. Available from: <<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5765>>. 45, 46, 47

CLOUTIER, R.; MULLER, G.; VERMA, D.; NILCHIANI, R.; HOLE, E.; BONE, M. The concept of reference architectures. **Systems Engineering**, v. 13, n. 1, p. 14–27, feb. 2010. ISSN 1098-1241. Available from: <<http://dx.doi.org/10.1002/sys.v13:1>>. 3, 31

COM: Component Object Model Technologies. 2017. Available from: <https://www.microsoft.com/com/default.aspx>. (Access in: 30 Oct. 2017). 35

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS (CCSDS). **Spacecraft monitor and control-core services, draft recommendation for space data system standards, CCSDS-522.0-R-2, proposed red book**. [S.l.], Abr 2008. 121 p. 64, 65

\_\_\_\_\_. **Mission operations common object model, draft recommendation for space data system standards, CCSDS-521.1-R-1, proposed red book**. [S.l.], Abr 2009. 64, 65

\_\_\_\_\_. **Mission operations reference model, recommended practice, CCSDS 520.1-M, magenta book.** [S.l.], Jul 2010. 71 p. [63](#), [65](#)

\_\_\_\_\_. **Mission operations services concept, informational report, CCSDS-520.0-G-3, green book.** [S.l.], Dec 2010. 62 p. [63](#), [64](#), [65](#)

CORBA component model specification version 4.0. 2016. Available from: <http://www.omg.org/spec/CCM/>. (Access in: 30 Oct. 2017). [35](#)

COUNCIL, N. R. **Technology for small spacecraft.** Washington, DC: The National Academies Press, 1994. ISBN 978-0-309-05075-3. Available from: <https://www.nap.edu/catalog/2351/technology-for-small-spacecraft>>. [8](#), [12](#)

CRNKOVIC, I.; CHAUDRON, M.; LARSSON, S. Component-based development process and component lifecycle. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ADVANCES, 2006. **Proceedings...** [S.l.], 2006. p. 44–44. [33](#)

CRNKOVIC, I.; SENTILLES, S.; VULGARAKIS, A.; CHAUDRON, M. R. V. A classification framework for software component models. **IEEE Transactions on Software Engineering**, v. 37, n. 5, p. 593–615, Sept 2011. ISSN 0098-5589. [34](#), [35](#), [36](#), [37](#), [88](#)

DAVID, L. **Cubesats: tiny spacecraft, huge payoffs.** Set 2004. Available from: <https://www.space.com/308-cubesats-tiny-spacecraft-huge-payoffs.html>>. Access in: 24 Nov. 2017. [11](#)

DOBRICA, L.; NIEMELÄ, E. An approach to reference architecture design for different domains of embedded systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH AND PRACTICE, 2008. **Proceedings...** [S.l.], 2008. p. 287–293. [31](#)

DOWLING, J.; CAHILL, V. Self-managed decentralised systems using k-components and collaborative reinforcement learning. In: ACM SIGSOFT WORKSHOP ON SELF-MANAGED SYSTEMS, 1., 2004. **Proceedings...** New York: ACM, 2004. p. 39–43. ISBN 1-58113-989-6. Available from: <http://doi.acm.org/10.1145/1075405.1075413>>. [35](#)

D'SOUZA, D. F.; WILLS, A. C. **Objects, components, and frameworks with UML: the catalysis approach**. Boston, MA, USA: Addison-Wesley Longman Publishing, 1999. ISBN 0-201-31012-0. 39

DUBOS, G. F.; CASTET, J.-F.; SALEH, J. H. Statistical reliability analysis of satellites by mass category: does spacecraft size matter? **Acta Astronautica**, v. 67, n. 5, p. 584 – 595, 2010. ISSN 0094-5765. Available from: <<http://www.sciencedirect.com/science/article/pii/S0094576510001347>>. 9

DURO, N.; MOREIRA, F.; ROGADO, J.; REIS, J.; PECCIA, N. Technology harmonization - developing a reference architecture for the ground segment software. In: IEEE AEROSPACE CONFERENCE, 2005. **Proceedings...** [S.l.], 2005. p. 3968–3979. ISSN 1095-323X. 3, 41, 42

DVORAK, D. NASA study on flight software complexity. In: AIAA INFOTECH AEROSPACE CONFERENCE, 2009. **Proceedings...** AIAA, 2009. Available from: <<https://doi.org/10.2514/6.2009-1882>>. 2

ENTERPRISE JavaBeans Technology. 2017. Available from: <<http://www.oracle.com/technetwork/java/javaee/ejb/index.html>>. Access in: 30 Oct. 2017. 35

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-E-ST-70C: space engineering – ground systems and operations – part 1: principles and requirements**. [S.l.], 2000. 8, 13, 15, 63, 65, 73

\_\_\_\_\_. **ECSS-E-10 Part 1B - system engineering — part 1: requirements and process**. [S.l.], 2004. 85, 86

\_\_\_\_\_. **ECSS-E-ST-70-31C: space engineering – ground systems and operations – monitoring and control data**. [S.l.], 2008. 65

\_\_\_\_\_. **ECSS-E-ST-40: space engineering – software general requirements**. Noordwijk, The Netherlands, Mar 2009. 64, 65

\_\_\_\_\_. **ECSS-E-ST-40C - software**. Noordwijk, The Netherlands, Mar 2009. 86

EUROPEAN SPACE AGENCY (ESA). **EGS-CC- european ground systems common core**. Oct. 2017. Available from: <http://www.egscc.esa.int/>. (Access in: 02 Nov. 2017). 42, 45

\_\_\_\_\_. **Global educational network for satellite operations / education / ESA**. Nov 2017. Available from: [http://www.esa.int/Education/Global\\_](http://www.esa.int/Education/Global_)



[Educational\\_Network\\_for\\_Satellite\\_Operations](#). ( Access in: 25 Nov. 2017).  
56, 57

FASSINO, J.-P.; STEFANI, J.-B.; LAWALL, J. L.; MULLER, G. Think: a software framework for component-based operating system kernels. In: GENERAL TRACK OF THE ANNUAL CONFERENCE ON USENIX ANNUAL TECHNICAL CONFERENCE, 2002. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2002. p. 73–86. ISBN 1-880446-00-6. Available from: <http://dl.acm.org/citation.cfm?id=647057.713860>>. 35

FERREIRA, M. G. V. **Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao software de controle de satélites**. 244 p. PhD Thesis (Doctor in Applied Computation) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2001-03-23 2001. Available from: <http://urlib.net/dpi.inpe.br/lise/2003/01.16.09.56>>. Access in: 24 nov. 2017. 15

FERRO, E.; GIROLAMI, M.; SALVI, D.; MAYER, C.; GORMAN, J.; GRGURIC, A.; RAM, R.; SADAT, R. The UniversAAL platform for AAL (Ambient Assisted Living). *Journal of Intelligent Systems*, p. 301–319, 2015. 3, 29

FILHO, N. F. D.; BARBOSA, E. F. A Contribution to the establishment of reference architectures for mobile learning environments. **IEEE Revista Iberoamericana de Tecnologias del Aprendizaje**, v. 10, n. 4, p. 234–241, Nov 2015. ISSN 1932-8540. 3, 29

FORD, B.; BACK, G.; BENSON, G.; LEPREAU, J.; LIN, A.; SHIVERS, O. The flux oskit: a substrate for kernel and language research. **SIGOPS - Operating Systems Review**, v. 31, n. 5, p. 38–51, oct. 1997. ISSN 0163-5980. Available from: <http://doi.acm.org/10.1145/269005.266642>>. 35

FORTESCUE, P.; STARK, J.; SWINERD, G. **Spacecraft systems engineering**. New York: Wiley, 2003. ISBN 9780470851029. Available from: <http://onlinelibrary.wiley.com/book/10.1002/9781119971009>>. 9

FURMENTO, N.; MAYER, A.; MCGOUGH, S.; NEWHOUSE, S.; FIELD, T.; DARLINGTON, J. Optimisation of component-based applications within a grid environment. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2001. **Proceedings...** New York: ACM, 2001. p. 30–30. ISBN 1-58113-293-X. Available from: <http://doi.acm.org/10.1145/582034.582064>>. 35

GALLAGHER, B. **Using the architecture tradeoff analysis method to evaluate a reference architecture: a case study**. Pittsburgh, PA, 2000. Available from:  
<<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5109>>. 31, 32

GALSKI, R. L. Capacidades tecnológicas do inpe em rastreo e controle de satélites: diagnóstico da situação atual e tendências de curto e médio prazo. In: MOREIRA, M. L. (Ed.). **Coletânea do I curso de pós-graduação em gestão estratégica da ciência e tecnologia em institutos públicos de pesquisa**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2012. Available from:  
<<http://urlib.net/sid.inpe.br/mtc-m19/2011/04.15.11.27>>. Access in: 24 nov. 2017. 15

GARLAN, D. Software architecture: a roadmap. In: CONFERENCE ON THE FUTURE OF SOFTWARE ENGINEERING, 2000. **Proceedings...** New York, NY, USA: ACM, 2000. p. 91–101. ISBN 1-58113-253-0. Available from:  
<<http://doi.acm.org/10.1145/336512.336537>>. 17

GARLAN, D.; ALLEN, R.; OCKERBLOOM, J. Architectural mismatch: why reuse is still so hard. **IEEE Software**, v. 26, n. 4, p. 66–69, July 2009. ISSN 0740-7459. 35, 36

GARLAN, D.; BACHMANN, F.; IVERS, J.; STAFFORD, J.; BASS, L.; CLEMENTS, P.; MERSON, P. **Documenting software architectures: views and beyond**. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321552687, 9780321552686. 19, 23

GÉDÉON, W. **OSGi and apache felix 3.0 beginners guide : build your very own OSGi applications using the flexible and powerful felix framework**. Birmingham, U.K: Packt Pub, 2010. ISBN 978-1-849511-38-4. 38

GENSSLER, T.; CHRISTOPH, A.; WINTER, M.; NIERSTRASZ, O.; DUCASSE, S.; WUYTS, R.; ARÉVALO, G.; SCHÖNHAGE, B.; MÜLLER, P.; STICH, C. Components for embedded software: the pecos approach. In: INTERNATIONAL CONFERENCE ON COMPILERS, ARCHITECTURE, AND SYNTHESIS FOR EMBEDDED SYSTEMS, 2002. **Proceedings...** New York, NY, USA: ACM, 2002. p. 19–26. ISBN 1-58113-575-0. Available from:  
<<http://doi.acm.org/10.1145/581630.581634>>. 35

- GMV. **hifly satellite control system**. Nov 2017. Available from:  
<https://www.gmv.com/en/Products/hifly/>. (Access in: 21 Nov. 2017.). 48, 50
- GOETZELMANN, M.; TUCKER, L.; MECREDY, N.; SANMARTI, J. The design of the european ground systems - common core (egs-cc). In: SPACEOPS CONFERENCE, 2014. **Proceedings...** AIAA, 2014. Available from:  
<https://doi.org/10.2514/6.2014-1768>>. 43, 44
- GRAAF, B.; DIJK, H. van; DEURSEN, A. van. Evaluating an Embedded Software Reference Architecture - Industrial Experience Report. In: EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING, 9., 2005. **Proceedings...** [S.l.], 2005. p. 354–363. ISSN 1534-5351. 31, 32
- GRACIANO NETO, V. V.; GARCÉS, L.; GUESSI, M.; OLIVEIRA, L. B. R. de; OQUENDO, F. On the equivalence between reference architectures and metamodels. In: INTERNATIONAL WORKSHOP ON EXPLORING COMPONENT-BASED TECHNIQUES FOR CONSTRUCTING REFERENCE ARCHITECTURES, 1., 2015. **Proceedings...** New York, NY, USA, 2015. p. 21–24. ISBN 978-1-4503-3445-7. Available from:  
<http://doi.acm.org/10.1145/2755567.2755572>>. 96
- GRIFFITH, R.; KAISER, G. Manipulating managed execution runtimes to support self-healing systems. **SIGSOFT - Software Engineering Notes**, v. 30, n. 4, p. 1–7, may 2005. ISSN 0163-5948. Available from:  
<http://doi.acm.org/10.1145/1082983.1083066>>. 35
- GUDMUNDSSON, V.; SCHULZE, C.; GANESAN, D.; LINDVALL, M.; WIEGAND, R. Model-based testing of NASA’s GMSEC, a reusable framework for ground system software. **Innovations in Systems and Software Engineering (ISSE)**, v. 11, n. 3, p. 217–232, 2015. 52
- GUESSI, M.; BUENO, L.; OLIVEIRA, R.; NAKAGAWA, E. Y. Representation of reference architectures: a systematic review. In: SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 2011. **Proceedings...** Montreal: ACM, 2011. p. 782–785. ISBN 1891706292. 72
- GUESSI, M.; OLIVEIRA, L. B. R.; GARCÉS, L.; OQUENDO, F. Towards a formal description of reference architectures for embedded systems. In: INTERNATIONAL WORKSHOP ON EXPLORING COMPONENT-BASED TECHNIQUES FOR CONSTRUCTING REFERENCE ARCHITECTURES, 1., 2015. **Proceedings...** New York, NY, USA: ACM, 2015. p. 17–20. ISBN

978-1-4503-3445-7. Available from:

<<http://doi.acm.org/10.1145/2755567.2755571>>. 3, 29

HANDY, M. **NASA technical reports server (NTRS) - GMSEC interface specification document**. [S.l.], Mar 2016. Available from: <<https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160005641.pdf>>.

Access in: (22 nov. 2017). 53, 55

HANSSON, H.; AKERHOLM, M.; CRNKOVIC, I.; TORNGREN, M. Saveccm - a component model for safety-critical real-time systems. In: EUROMICRO CONFERENCE, 30., 2004. **Proceedings...** Washington, DC, USA: IEEE, 2004. p. 627–635. ISBN 0-7695-2199-1. Available from:

<<https://doi.org/10.1109/EUROMICRO.2004.72>>. 35

HEIDT, M. H.; PUIG-SUARI, P. J.; AUGUSTUS, P.; MOORE, S.; NAKASUKA, P. S.; ROBERT, P.; TWIGGS, J. Cubesat: a new generation of picosatellite for education and industry low-cost space experimentation. In: ANNUAL USU CONFERENCE ON SMALL SATELLITES, 14., 2000. **Proceedings...** [S.l.], 2000. 11, 12

HEINEMAN, G. T.; COUNCILL, W. T. (Ed.). **Component-based software engineering: putting the pieces together**. Boston, MA, USA: Addison-Wesley Longman Publishing, 2001. ISBN 0-201-70485-4. 33, 88

HELANDER, J.; FORIN, A. Mmlite: a highly componentized system architecture. In: ACM SIGOPS EUROPEAN WORKSHOP ON SUPPORT FOR COMPOSING DISTRIBUTED APPLICATIONS, 8., 1998. **Proceedings...** New York, NY, USA: ACM, 1998. p. 96–103. Available from:

<<http://doi.acm.org/10.1145/319195.319210>>. 35

HELVAJIAN, H.; JANSON, S. **Small satellites: past, present, and future**. Aerospace Press, 2008. ISBN 9781884989223. Available from:

<<https://books.google.com.br/books?id=aJMsPAAACAAJ>>. 9

HERZUM, P.; SIMS, O. **Business components factory: a comprehensive overview of component-based development for the enterprise**. New York, NY, USA: John Wiley & Sons, 2000. ISBN 0471327603. 24

HOFMEISTER, C.; NORD, R. L.; SONI, D. Describing software architecture with uml. In: WORKIN IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 1., 1999. **Proceedings...** Deventer, The Netherlands, The Netherlands: Kluwer, 1999.

p. 145–160. ISBN 0-7923-8453-9. Available from:

<<http://dl.acm.org/citation.cfm?id=646545.696368>>. 24

INSTITUTO FOR ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE**

**Recommended practice for architectural description of**

**software-intensive systems**. [S.l.], 2000. i–23 p. Available from:

<<http://dx.doi.org/10.1109/ieeestd.2000.91944>>. 26, 64, 73

\_\_\_\_\_. **Systems and software engineering - recommended practice for architectural description of software-intensive systems (ISO/IEC 42010 IEEE Std 1471-2000)**. [S.l.], jul. 2007. c1–24 p. Available from:

<<http://dx.doi.org/10.1109/ieeestd.2007.386501>>. 17, 25, 72

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **CRC - Centro de Rastreio e Controle de Satélites**. nov 2017. Available from:

<<http://www.inpe.br/crc/>>. Access in: 29 nov. 2017. 17

IVERS, J.; CLEMENTS, P.; GARLAN, D.; NORD, R.; SCHMERL, B.; SILVA, O. **Documenting component and connector views with UML 2.0**. Pittsburgh, PA, 2004. Available from:

<<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7095>>.

25

JOHNSON, E. J.; KUNZE, A. R. **Ixp2400-2800 programming: the complete microengine coding guide**. [S.l.]: Intel Press, 2003. ISBN 097178616X. 35

JONES, T. C. **Programming productivity : steps toward a science**. New York, United States: McGraw-Hill, 1986. 276 p. ISSN 0070328110. 3

KANG, K. C.; KIM, S.; LEE, J.; KIM, K.; SHIN, E.; HUH, M. Form: a feature-oriented reuse method with domain-specific reference architectures. **Annals of Software Engineering**, v. 5, n. 1, p. 143–168, jan. 1998. ISSN 1022-7091. Available from:

<<http://dl.acm.org/citation.cfm?id=590631.590645>>. 39

KARLIN, S.; PETERSON, L. Vera: an extensible router architecture. In: **IEEE OPEN ARCHITECTURES AND NETWORK PROGRAMMING**, 2001.

**Proceedings...** [S.l.]: IEEE, 2001. p. 3–14. 35

KLEIN, J.; BUGLAK, R.; BLOCKOW, D.; WUTTKE, T.; COOPER, B. A reference architecture for big data systems in the national security domain. In: **INTERNATIONAL WORKSHOP ON BIG DATA SOFTWARE ENGINEERING**,

- 2., 2016. **Proceedings...** New York, NY, USA: ACM, 2016. p. 51–57. ISBN 978-1-4503-4152-3. Available from:  
<<http://doi.acm.org/10.1145/2896825.2896834>>. 3, 29
- KOTONYA, G.; ONYINO, W.; HUTCHINSON, J.; SAWYER, P.; CANAL, J. Cots component-based system development. In: \_\_\_\_\_. **Business component-based software engineering**. Boston, MA: Springer US, 2003. p. 227–245. ISBN 978-1-4615-1175-5. Available from:  
<[https://doi.org/10.1007/978-1-4615-1175-5\\_13](https://doi.org/10.1007/978-1-4615-1175-5_13)>. 38
- KOZACZYNSKI, W. **Composite nature of component**. Los Angeles, EUA: Internacional Workshop on ComponentBased Software Engineering, May 1999. 33
- KRUCHTEN, P. The 4+1 view model of architecture. **IEEE Software**, IEEE, v. 12, n. 6, p. 42–50, nov. 1995. ISSN 0740-7459. Available from:  
<<http://dx.doi.org/10.1109/52.469759>>. 23, 41
- \_\_\_\_\_. **The rational unified process: an introduction**. 2. ed. Boston, MA, USA: Addison-Wesley Longman Publishing, 2000. ISBN 0201707101. 27
- KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The past, present, and future for software architecture. **IEEE Software**, v. 23, n. 2, p. 22–30, mar. 2006. ISSN 0740-7459. Available from: <<http://dx.doi.org/10.1109/MS.2006.59>>. 27, 28
- KRUEGER, C. W. Software reuse. **ACM Computing Surveys**, v. 24, n. 2, p. 131–183, jun. 1992. ISSN 0360-0300. Available from:  
<<http://doi.acm.org/10.1145/130844.130856>>. 38
- KRUGER, I. H.; MATHEW, R. Systematic development and exploration of service-oriented software architectures. In: WORKING IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 4., 2004. **Proceedings...** [S.l.], 2004. p. 177–187. 18
- LARMAN, C. **Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development**. Upper Saddle River, N.J: Prentice Hall PTR, 2005. ISBN 0131489062. 21
- LAU, K.-K.; WANG, Z. A taxonomy of software component models. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 2005. **Proceedings...** 2005. p. 88–95. ISBN 0-7695-2431-1. Available from:  
<<http://dx.doi.org/10.1109/EUROMICRO.2005.8>>. 88

LAU, K. K.; WANG, Z. Software component models. **IEEE Transactions on Software Engineering**, v. 33, n. 10, p. 709–724, Oct 2007. ISSN 0098-5589. 88, 95

LEVEQUE, K.; PUIG-SUARI, J.; TURNER, C. Global educational network for satellite operations (genso). **ANNUAL AIAA USU CONFERENCE ON SMALL SATELLITES**, 21., 2007, San Luis Obispo: California Polytechnic State University, 2007. Available from: <<https://digitalcommons.usu.edu/smallsat/2007/all2007/73/>>. 56, 57, 58

LI, J.; GUPTA, A.; ARVID, J.; BORRETZEN, B.; CONRADI, R. The empirical studies on quality benefits of reusing software components. In: **ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE**, 31., 2007. **Proceedings...** [S.l.], 2007. v. 2, p. 399–402. ISSN 0730-3157. 95

LOPEZ, T.; FRAGA, E. hifly anywhere remote web operations. In: **INTERNATIONAL CONFERENCE ON SPACE OPERATIONS**, 2012. **Proceedings...** Stockholm, Sweden, 2012. ISBN 978-1-62993-433-4. Available from: <<http://www.spaceops2012.org/proceedings/documents/id1294607-Paper-002.pdf>>. 48, 49

LOWE, C.; MACDONALD, M. Rapid model-based inter-disciplinary design of a cubesat mission. **Acta Astronautica**, v. 105, n. 1, p. 321–332, 12 2014. ISSN 0094-5765. Date of Acceptance: 02/10/2014. 10, 11

MAGOUTIS, K.; BRUSTOLONI, J. C.; GABBER, E.; NG, W. T.; SILBERSCHATZ, A. Building appliances out of components using pebble. In: **WORKSHOP ON ACEM SIGOPS EUROPEAN WORKSHOP: BEYOND THE PC: NEW CHALLENGES FOR THE OPERATING SYSTEM**, 9., 2000. **Proceedings...** New York, NY, USA: ACM, 2000. p. 211–216. Available from: <<http://doi.acm.org/10.1145/566726.566769>>. 35

MARTÍNEZ-FERNÁNDEZ, S. Towards supporting the adoption of software reference architectures: an empirically-grounded framework. In: **INTERNATIONAL DOCTORAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING**, 2013. **Proceedings...** 2013. p. 1–8. Available from: <<http://hdl.handle.net/2117/21144>; <http://umbc.edu/eseiw2013/idoese/program.shtml>>. 3, 28



MARTÍNEZ-FERNÁNDEZ, S.; AYALA, C. P.; FRANCH, X.; MARQUES, H. M. Rearm: a reuse-based economic model for software reference architectures. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE, 13., 2013. **Proceedings...** Berlin, Heidelberg, 2013. p. 97–112. ISBN 978-3-642-38977-1. Available from: <[https://doi.org/10.1007/978-3-642-38977-1\\_7](https://doi.org/10.1007/978-3-642-38977-1_7)>. 95

MARTÍNEZ-FERNÁNDEZ, S.; AYALA, C. P.; FRANCH, X.; NAKAGAWA, E. Y. A survey on the benefits and drawbacks of AUTOSAR. In: INTERNATIONAL WORKSHOP ON AUTOMOTIVE SOFTWARE ARCHITECTURE, 1., 2015. **Proceedings...** [S.l.], 2015. p. 19–26. 3, 29

MAYORGA, A. An auto configuration system for the GMSEC architecture and API. In: IEEE INTERNATIONAL CONFERENCE ON SPACE MISSION CHALLENGES FOR INFORMATION TECHNOLOGY, 2., 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 1–485. 52

MCAFFER, J.; VANDERLEI, P.; ARCHER, S. **OSGi and equinox: creating highly modular java systems**. Upper Saddle River, NJ: Addison-Wesley, 2010. (Eclipse Series). ISBN 978-0-321-58571-4. Available from: <<https://www.safaribooksonline.com/library/view/osgi-and-equinox/9780321561510/>>. 38

MCCLURE, C. **Software reuse techniques: adding reuse to the system development process**. Upper Saddle River, NJ, USA: Prentice-Hall, 1997. ISBN 0-13-661000-5. 33, 38

MICROSOFT. **NET: architectural components** | Microsoft Docs. 2017. Available from: <<https://docs.microsoft.com/en-us/dotnet/standard/components>>. Access in: 30 Oct. 2017. 35

MILI, H.; MILI, F.; MILI, A. Reusing software: issues and research directions. **IEEE Transactions on Software Engineering**, v. 21, n. 6, p. 528–562, Jun 1995. ISSN 0098-5589. 95

MONROE, R. T.; KOMPANEK, A.; MELTON, R.; GARLAN, D. Architectural styles, design patterns, and objects. **IEEE Software**, v. 14, n. 1, p. 43–52, jan. 1997. ISSN 0740-7459. Available from: <<http://dx.doi.org/10.1109/52.566427>>. 18



MOREL, T.; LOPEZ, T.; CASAS, N. Deploying operational multi-satellite control centres on virtual environments. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS, 2014. **Proceedings...** Pasadena, CA, 2014. 47, 50

NAKAGAWA, E. Y.; ANTONINO, P. O.; BECKER, M. Reference architecture and product line architecture: a subtle but critical difference. In: EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE, 5., 2011. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2011. p. 207–211. ISBN 978-3-642-23797-3. Available from: <<http://dl.acm.org/citation.cfm?id=2041790.2041818>>. 3, 28, 29, 30

NAKAGAWA, E. Y.; BECKER, M.; MALDONADO, J. C. Towards a process to design product line architectures based on reference architectures. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 17., 2013. **Proceedings...** New York, NY, USA: ACM, 2013. p. 157–161. ISBN 978-1-4503-1968-3. Available from: <<http://doi.acm.org/10.1145/2491627.2491651>>. 27, 28

NAKAGAWA, E. Y.; FERRARI, F. C.; SASAKI, M. M. F.; MALDONADO, J. C. An aspect-oriented reference architecture for software engineering environments. **Journal of Systems and Software**, v. 84, n. 10, p. 1670–1684, oct. 2011. ISSN 0164-1212. Available from: <<http://dx.doi.org/10.1016/j.jss.2011.04.052>>. 3, 29

NAKAGAWA, E. Y.; GUESSI, M.; MALDONADO, J. C.; FEITOSA, D.; OQUENDO, F. Consolidating a process for the design, representation, and evaluation of reference architectures. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2014. **Proceedings...** [S.l.], 2014. p. 143–152. 4, 31, 63

NAKAGAWA, E. Y.; MALDONADO, J. C. Reference architecture knowledge representation: an experience. In: INTERNATIONAL WORKSHOP ON SHARING AND REUSING ARCHITECTURAL KNOWLEDGE, 3., 2008. **Proceedings...** 2008. p. 51–54. ISBN 978-1-60558-038-8. Available from: <<http://doi.acm.org/10.1145/1370062.1370077>>. 72

NAKAGAWA, E. Y.; OQUENDO, F.; BECKER, M. RAModel: A reference model for reference architectures. In: JOINT WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE AND EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE, 2012. **Proceedings...** [S.l.]: IEEE, 2012. p. 297–301. 31, 32, 33, 81

NAKAGAWA, E. Y.; OQUENDO, F.; MALDONADO, J. C. Reference architectures. In: **OUSSALAH, M. C. (Ed.). Software Architecture 1**. New York: John Wiley & Sons, 2014. p. 55–82. Available from: <https://doi.org/10.1002%2F9781118930960.ch2>>. 29

NASA. **NASA systems engineering handbook revision 2**. [S.l.], 2007. Available from: <https://www.nasa.gov/connect/ebooks/nasa-systems-engineering-handbook>>. 85

NASA. **NASA mission operations and communications services**. [S.l.], 2014. Available from: [https://deepspace.jpl.nasa.gov/files/dsn/6\\_NASA\\_MOCS\\_2014\\_10\\_01\\_14.pdf](https://deepspace.jpl.nasa.gov/files/dsn/6_NASA_MOCS_2014_10_01_14.pdf)>. 2

\_\_\_\_\_. **Deep space network services catalog**. [S.l.], 2015. Available from: <https://deepspace.jpl.nasa.gov/files/dsn/820-100-F1.pdf>>. 2

NASA. **Small spacecraft technology state of the art**. [S.l.], 2015. Available from: [https://www.nasa.gov/sites/default/files/atoms/files/state\\_of\\_the\\_art-aug2016.pdf](https://www.nasa.gov/sites/default/files/atoms/files/state_of_the_art-aug2016.pdf)>. 9

NOGUERO, J.; JULIAN, G. G.; BEECH, T. W. Mission control system for earth observation missions based on SCOS-2000. In: IEEE AEROSPACE CONFERENCE, 2005. **Proceedings...** [S.l.]: IEEE, 2005. p. 4088–4099. ISSN 1095-323X. 51

OMMERING, R. van; LINDEN, F. van der; KRAMER, J.; MAGEE, J. The koala component model for consumer electronics software. **Computer**, v. 33, n. 3, p. 78–85, mar. 2000. ISSN 0018-9162. Available from: <http://dx.doi.org/10.1109/2.825699>>. 35

OPENGROUP. **ArchiMate 3.0.1 specification**. Nov 2017. Available from: <http://pubs.opengroup.org/architecture/archimate3-doc/toc.html>>. Access in: 30 Oct. 2017. 27

OSGI™ Alliance - Architecture. Oct 2017. Available from: <https://www.osgi.org/developer/architecture>>. Access in: 30 Oct. 2017. 35, 37

OSORIO, R. V.; LEMOS, J. P.; BEECH, T. W.; JULIAN, G. G.; CHAUMON, J. P. SCOS-2000 release 4.0 : Multi-mission/multi-domain capabilities in ESA SCOS-2000 MCS kernel, 2006. In: IEEE AEROSPACE CONFERENCE, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 1–17. ISSN 1095-323X. 51, 52

OUSSALAH, M. **Software architecture 2**. [S.l.]: Wiley, 2014. (Computer engineering series). ISBN 9781118945100. 3, 27, 28

PANUNZIO, M. **Definition, realization and evaluation of a software reference architecture for use in space applications**. PhD Thesis (PhD) — University of Bologna, Italy, 2011. 3, 29, 88

PANUNZIO, M.; VARDANEGA, T. A component model for on-board software applications. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 36., 2010. **Proceedings...** [S.l.], 2010. p. 57–64. ISSN 1089-6503. 35, 88

\_\_\_\_\_. On software reference architectures and their application to the space domain. In: FAVARO, J. M.; MORISIO, M. (Ed.). **Safe and Secure Software Reuse**. Springer, 2013. (Lecture Notes in Computer Science, v. 7925), p. 144–159. ISBN 978-3-642-38976-4. Available from:  
<[https://doi.org/10.1007/978-3-642-38977-1\\_10](https://doi.org/10.1007/978-3-642-38977-1_10)>. 3, 42

PARDAVILA, R. T. **SATNet project Documentation**. Nov 2013. Available from: <<https://github.com/satnet-project/documentation>>. Access in: 25 Nov. 2017. 59, 60

PARDAVILA, R. T.; ESPINDOLA, J. E. D.; ROHLING, A. J.; FERREIRA, M. G. V.; SANTOS, W. A.; PUIG-SUARI, J.; AGUADO, A. F. Integration of the INPE ground station into the satnet network for supporting small satellites programs in brazil. In: LATIN AMERICAN IAA CUBESAT WORKSHOP, 2., 2016. **Proceedings...** Florianopolis, Brazil, 2016. 59, 60, 61, 62

PARDAVILA, R. T.; VAZQUEZ, A.; PUIG, J.; KURAHARA, N.; BELLARDO, J. Towards an open-source ground stations network for cubesats. In: ANNUAL CUBESAT DEVELOPERS WORKSHOP, 11., 2014. **Proceedings...** San Luis Obispo: California Polytechnic State University, 2014. Available from: <<http://mstl.atl.calpoly.edu/~bklofas/Presentations/DevelopersWorkshop2014/>>. 58, 62

PECCHIOLI, M.; CARRANZA, J. M. The main concepts of the european ground systems – common core (EGS-CC). In: GSAW-GROUND SYSTEM ARCHITECTURES WORKSHOP. **Proceedings...** 2013. Available from: <<http://gsaw.org/past-proceedings/2013-2/>>. Access in: 02 Nov. 2017. 43, 44

\_\_\_\_\_. Highlights of the european ground system – common core initiative. In: INTERNATIONAL CONFERENCE ON ACCELERATOR AND LARGE EXPERIMENTAL PHYSICS CONTROL SYSTEMS, 2017. **Proceedings...** 2017. Available from: <<http://icalepcs2017.vrws.de>>. Access in: 02 Nov. 2017. 44

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. **Software Engineering Notes.**, v. 17, n. 4, p. 40–52, oct. 1992. ISSN 0163-5948. Available from: <<http://doi.acm.org/10.1145/141874.141884>>. 18

PISACANE, V. **Fundamentals of space systems.** [S.l.]: Oxford University Press, 2005. (Applied Physics Laboratory series in science and engineering). ISBN 9780195162059. 7

POUR, G. Component-based software development approach: new opportunities and challenges. In: TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES TOOLS, 1998. **Proceedings...** [S.l.], 1998. p. 376–383. 38

PRESSMAN, R. **Software engineering : a practitioner's approach.** Boston, Mass: McGraw Hill, 2001. ISBN 0-07-365578-3. 38, 39

PUIG-SUARI, J.; TURNER, C.; AHLGREN, W. Development of the standard cubesat deployer and a cubesat class picosatellite. In: IEEE AEROSPACE CONFERENCE, 2011. **Proceedings...** [S.l.]: IEEE, 2001. p. 1/347–1/353 vol.1. 8, 10

REID, S. European technology harmonisation on ground software systems: reference architecture and ICDs. In: SPACEOPS CONFERENCE, 2012. **Proceedings...** AIAA, 2012. Available from: <<https://doi.org/10.2514%2F6.2012-1295675>>. 41

RODRÍGUEZ, L. M. G.; AMPATZOGLOU, A.; AVGERIOU, P.; NAKAGAWA, E. Y. A reference architecture for healthcare supportive home systems. In: INTERNATIONAL SYMPOSIUM ON COMPUTER-BASED MEDICAL SYSTEMS, 28., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 358–359. ISSN 1063-7125. 29

ROMAN, M.; MICKUNAS, M. D.; KON, F.; CAMPBELL, R. Legorb and ubiquitous CORBA. In: IFIP/ACM WORKSHOP ON REFLECTIVE MIDDLEWARE, 2000. **Proceedings...** [S.l.]: ACM, 2000. p. 1–2. 35

ROZANSKI, N.; WOODS, E. **Software systems architecture: working with stakeholders using viewpoints and perspectives**. [S.l.]: Addison-Wesley Professional, 2005. ISBN 0321112296. 41

ROZENFELD, P.; ORLANDO, V.; FERREIRA, M. Applying the 21st century technology to the 20th century mission control. In: SPACEOPS CONFERENCE, 2002. **Proceedings...** AIAA, 2002. Available from: <<https://doi.org/10.2514/2F6.2002-t2-78>>. 15, 17

SAMETINGER, J. **Software engineering with reusable components**. New York, USA: Springer-Verlag, 1997. ISBN 3-540-62695-6. 33

SANTOS, J. F. M.; GUESSI, M.; GALSTER, M.; FEITOSA, D.; NAKAGAWA, E. Y. A checklist for evaluation of reference architectures of embedded systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 25., 2013. **Proceedings...** 2013. p. 451–454. Available from: <<http://dblp.org/rec/bib/conf/seke/SantosGGFN13>>. 31, 32

SCHOLZ, A.; JUANG, J.-N. Toward open source cubesat design. **Acta Astronautica**, v. 115, n. Supplement C, p. 384 – 392, 2015. ISSN 0094-5765. Available from: <<http://www.sciencedirect.com/science/article/pii/S0094576515002507>>. 10

SHAMES, P.; YAMADA, T. Reference architecture for space data systems. In: INTERNATIONAL SYMPOSIUM ON REDUCING THE COST OF SPACECRAFT GROUND SYSTEMS AND OPERATIONS, 5., 2003. **Proceedings...** Jet Propulsion Laboratory, 2003. p. 8–11. Available from: <<https://descanso.jpl.nasa.gov/RCSGS0/program.html>>. Access in: 03 Oct. 2017. 2, 3, 41

\_\_\_\_\_. **Tools for Describing the Reference Architecture for Space Data Systems**. Jet Propulsion Laboratory, National Aeronautics and Space Administration Pasadena, CA, 05 2004. Available from: <<https://trs.jpl.nasa.gov/handle/2014/38436>>. Access in: 04 Nov. 2017. 2, 3

SHAW, M.; CLEMENTS, P. The golden age of software architecture. **IEEE Software**, v. 23, n. 2, p. 31–39, March 2006. ISSN 0740-7459. 28

SHAW, M.; GARLAN, D. **Software architecture: perspectives on an emerging discipline**. Upper Saddle River, NJ, USA: Prentice-Hall, 1996. ISBN 0-13-182957-2. 19, 27

SHETH, S.; ARORA, N.; MURPHY, C.; KAISER, G. The wehelp reference architecture for community-driven recommender systems. In: INTERNATIONAL WORKSHOP ON RECOMMENDATION SYSTEMS FOR SOFTWARE ENGINEERING, 2., 2010. **Proceedings...** New York, NY, USA: ACM, 2010. p. 46–47. ISBN 978-1-60558-974-9. Available from: <http://doi.acm.org/10.1145/1808920.1808930>>. 3, 29

SHIROMA, W. A.; MARTIN, L. K.; AKAGI, J. M.; AKAGI, J. T.; WOLFE, B. L.; FEWELL, B. A.; OHTA, A. T. Cubesats: a bright future for nanosatellites. **Central European Journal of Engineering**, v. 1, n. 1, p. 9–15, Mar 2011. ISSN 2081-9927. Available from: <https://doi.org/10.2478/s13531-011-0007-8>>. 9

SMITH, D.; BRISTOW, J.; WILMOT, J. A successful component architecture for interoperable and evolvable ground data systems. In: SPACEOPS CONFERENCE, 2006. **Proceedings...** AIAA, 2006. Available from: <https://doi.org/10.2514/2.F6.2006-5743>>. 52, 54

SMITH, D.; GRUBB, T.; ESPER, J. Linking and combining distributed operations facilities using NASA's GMSEC systems architectures. In: SPACEOPS CONFERENCE, 2008. **Proceedings...** [S.l.]: AIAA, 2008. 3

SORENSEN, T.; PILGER, E.; YOST, B.; NUNES, M.; DIFFERDING, J. Plug and play mission operations. In: IEEE AEROSPACE CONFERENCE, 2012. **Proceedings...** [S.l.]: IEEE, 2012. p. 1–13. ISSN 1095-323X. 2

SPACEWORKS ENTERPRISES. **2017 Nano/Microsatellite Market Forecast**. Atlanta, GA, 2017. Available from: [http://spaceworksforecast.com/docs/SpaceWorks\\_Nano\\_Microsatellite\\_Market\\_Forecast\\_2017.pdf](http://spaceworksforecast.com/docs/SpaceWorks_Nano_Microsatellite_Market_Forecast_2017.pdf)>. 1, 2

SPACEWORKS ENTERPRISES. **Small satellite report - trends and market observations**. [S.l.], 2017. Available from: <http://spaceworksforecast.com>>. Access in: 02 Out. 2017. 13, 14

SULLIVAN, T.; SATHER, D.; NISHINAGA, R. A flexible satellite command and control framework: developing responsive and agile space systems. In: **Crosslink® Magazine - The Aerospace Corporation**. Los Angeles, CA, USA: [s.n.], 2009. ISSN 1527-5264. Available from: <http://www.aerospace.org/crosslinkmag/summer-2009/a-flexible-satellite-command-and-control-framework>>. 3, 56

SUTHERLAND, B. **Modern warfare, intelligence and deterrence: the technologies that are transforming them**. Wiley, 2012. (The Economist). ISBN 9781118240441. Available from: <<https://profilebooks.com/the-economist-modern-warfare-intelligence-and-deterrence.html>>. 11

SWARTWOUT, M. **CubeSat database**. Nov 2017. Available from: <<https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database>>. Access in: 24 Nov. 2017). 10

SZYPERSKI, C. **Component software : beyond object-oriented programming**. London: Addison-Wesley, 2011. ISBN 978-0321753021. 38

TAYLOR, R. N.; MEDVIDOVIC, N.; ANDERSON, K. M.; WHITEHEAD, E. J.; ROBBINS, J. E.; NIES, K. A.; OREIZY, P.; DUBROW, D. L. A component- and message-based architectural style for gui software. **IEEE Transactions on Software Engineering**, v. 22, n. 6, p. 390–406, Jun 1996. ISSN 0098-5589. 20

THE CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS. **Cross support reference model — Part 1: space link extension services**. [S.l.], 2005. 95 p. p. 2, 64, 65

\_\_\_\_\_. **Cross support concept — Part 1: space link extension services CCSDS-910.3-G-3**. [S.l.], 2006. 100 p. p. 2

\_\_\_\_\_. **Reference architecture for space data systems, recommended practice, CCSDS 311.0-M-1**. Washington, D.C, USA, 2008. 3, 41, 64, 65, 72

\_\_\_\_\_. **Reference architecture for space information management, informational report**. Washington, D.C, USA, 2013. 3, 42, 65

THYAGARAJAN, K.; GUPTA, J.; GOEL, P.; JAYARAMAN, K. University small satellite program—anusat. **Acta Astronautica**, v. 56, n. 1, p. 89 – 97, 2005. ISSN 0094-5765. 4th IAA International Symposium on Small Satellites for Earth Observation. Available from: <<http://www.sciencedirect.com/science/article/pii/S0094576504002863>>. 11

TOMINAGA, J. **Simulador de satélites para verificação de planos de operações em voo**. 174 p. Master Thesis (Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010. Available from: <<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/05.24.18.55>>. Access in: 30 Nov. 2017. 16



TRACZ, W. Software reuse myths. **SIGSOFT Software Engineering Notes**, v. 13, n. 1, p. 17–21, jan. 1988. ISSN 0163-5948. Available from: <http://doi.acm.org/10.1145/43857.43859>>. 3

UML Unified Modeling Language, Specification V 2.5. Mar 2015. Available from: <http://www.omg.org/spec/UML/2.5>>. Access in: 05 Nov. 2017. 26

UNION OF CONCERNED SCIENTISTS (UCS). **UCS Satellite Database**. [S.l.], 2017. Available from: <http://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database>>. Access in: 03 Oct. 2017. 1

UNITED NATIONS OFFICE FOR OUTER SPACE AFFAIRS (UNOOSA). **Online index of objects launched into outer space**. 2017. Available from: <http://www.unoosa.org/oosa/osoindex/search-ng.jsp>>. Access in: 01 Nov. 2017. 1

WASSERMAN, A. I. Toward a discipline of software engineering. **IEEE Software**, v. 13, n. 6, p. 23–31, nov. 1996. ISSN 0740-7459. Available from: <http://dx.doi.org/10.1109/52.542291>>. 18

WENTZEL, K. D. Software reuse - facts and myths. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 16., 1994. **Proceedings...** Los Alamitos, CA, USA: IEEE, 1994. p. 267–268. ISBN 0-8186-5855-X. Available from: <http://dl.acm.org/citation.cfm?id=257734.257779>>. 3

WERTZ, J.; LARSON, W. **Space mission analysis and design**. [S.l.]: Springer, 1999. (Space Technology Library). ISBN 9780792359012. 2, 8

WEYRICH, M.; EBERT, C. Reference architectures for the internet of things. **IEEE Software**, v. 33, n. 1, p. 112–116, Jan 2016. ISSN 0740-7459. 3, 29

YACOUB, S.; AMMAR, H.; MILI, A. A model for classifying component interfaces. In: WORKSHOP ON COMPONENT-BASED SOFTWARE ENGINEERING, 1999. **Proceedings...** 1999. Available from: <https://www.bibsonomy.org/bibtex/25b53fc9d984583b54978ede7a4e51ad5/pbrada>>. 33

ZHU, H. **Software design methodology**. Oxford: Elsevier, 2005. ISBN 9780750660754. 18, 19, 20, 21, 22, 23



## **PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE**

### **Teses e Dissertações (TDI)**

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

### **Manuais Técnicos (MAN)**

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

### **Notas Técnico-Científicas (NTC)**

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

### **Relatórios de Pesquisa (RPQ)**

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

### **Propostas e Relatórios de Projetos (PRP)**

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

### **Publicações Didáticas (PUD)**

Incluem apostilas, notas de aula e manuais didáticos.

### **Publicações Seriadas**

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

### **Programas de Computador (PDC)**

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

### **Pré-publicações (PRE)**

Todos os artigos publicados em periódicos, anais e como capítulos de livros.