



**CISTER**

Research Centre in  
Real-Time & Embedded  
Computing Systems

# PhD Thesis

---

## **Integrated Method For Designing Complex Cyber-Physical Systems**

**Fernando Gonçalves**

---

CISTER-TR-181120

# Integrated Method For Designing Complex Cyber-Physical Systems

Fernando Gonçalves

\*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

<http://www.cister.isep.ipp.pt>

## Abstract

The design of a Cyber-Physical System (CPS) is defined as a complex activity, being composed of a set of design phases devoted to model application characteristics. In this sense, detailing the phase characteristics is required in order to help the design teams during the project design, and to aim the support of the correct application characteristics representation. However, despite some of these phases, such as the control systems design, being discussed at length by the engineering community, other phases have less detailed studies, i.e., the design activities that compose these phases are not so well documented. In these sense, it is required more experience from the design teams to perform activities such as, the sensing and actuation subsystems representation, and the integration of formal verification methods on the design process, among others. Despite the lack of information related to them, these phases are essential to the CPS design, by the fact that they support application characteristics representation and the properties validation, as well as, they also provide the integration between designed system and their environment. Regarding the CPS design process, different methods are available in the literature, aiming to guide the designers to perform the modeling tasks. However, these approaches do not provide enough information related to those described activities. In this context, this thesis proposes an integrated method applied to CPS design, more specifically devoted to the Unmanned Aerial Vehicles (UAV) design. That proposed method aims to integrate different modeling processes such as functional, architectural, sensor and actuator integrations, and formal verification design processes. Based on the proposed activities this method aims to support the UAV embedded system design, and allow the integration between the embedded platform and the set of system devices. The Model Driven Engineering (MDE) is used as basis to the proposed approach, and aims to support the automated model generation based on the application characteristics. It is intended to ensure the maintainability of the system information over all the design steps and provide the property evaluation and validation, considering the model transformation principles. Two different tools are designed with the proposed method, the ECPS Modeling and the ECPS Verifier, for supporting the design activities. The ECPS Modeling provides the transformation process from functional model to architectural model, in order to integrate sensor and actuator characteristics. On the other hand, the ECPS Verifier provides the CPS behavior representation, based on the architectural model, by using timed automatas, which allows the formal verification evaluation by performing model checking. The proposed method and the designed tools are applied on the project of a tilt-rotor UAV design. The details of the method proposed in this thesis are demonstrated by performing the UAV project, described as a case study.

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS**

Fernando Silvano Gonçalves

**INTEGRATED METHOD FOR DESIGNING COMPLEX  
CYBER-PHYSICAL SYSTEMS**

Florianópolis

2018



Fernando Silvano Gonçalves

**INTEGRATED METHOD FOR DESIGNING COMPLEX  
CYBER-PHYSICAL SYSTEMS**

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.

Orientador: Prof. Dr. Leandro Buss  
Becker - PGEAS - UFSC

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,  
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Gonçalves, Fernando Silvano  
Integrated Method for Designing Complex Cyber  
Physical Systems / Fernando Silvano Gonçalves ;  
orientador, Leandro Buss Becker, 2018.  
175 p.

Tese (doutorado) - Universidade Federal de Santa  
Catarina, Centro Tecnológico, Programa de Pós  
Graduação em Engenharia de Automação e Sistemas,  
Florianópolis, 2018.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Projeto  
de CPS. 3. VANT. 4. Engenharia dirigida por  
modelos. 5. Transformação de modelos. I. Becker,  
Leandro Buss. II. Universidade Federal de Santa  
Catarina. Programa de Pós-Graduação em Engenharia de  
Automação e Sistemas. III. Título.

Fernando Silvano Gonçalves

**INTEGRATED METHOD FOR DESIGNING COMPLEX  
CYBER-PHYSICAL SYSTEMS**

Esta Tese foi julgada aprovada para a obtenção do Título de “Doutor em Engenharia de Automação e Sistemas”, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 17 de julho 2018.



---

Prof. Dr. Werner Kraus Junior  
PGEAS - UFSC

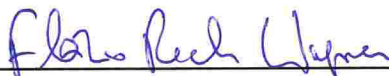
Coordenador do Programa de Pós-Graduação em Engenharia de  
Automação e Sistemas



---

Prof. Dr. Leandro Buss Becker - PGEAS - UFSC  
Orientador

**Banca Examinadora:**



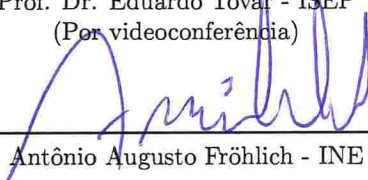
---

Prof. Dr. Flávio Rech Wagner - UFRGS



---

p/ Prof. Dr. Eduardo Tovar - ISEP  
(Por videoconferência)



---

Prof. Dr. Antônio Augusto Fröhlich - INE - UFSC



To my caring parents, Claudi and Juçara,  
to my sister, Daniele, and to my love,  
Katren.



## ACKNOWLEDGMENTS

First of all I would like to thank God, for the gift of life, for giving me wisdom and science, being with me constantly and guiding me in my choices.

I want to thank my lovely family, because without them none of this would be possible. Thank you for your love, attention, and understanding, as well as, for always encouraging and supporting me in my choices, contributing to my personal and professional growth. I give special thanks to my love Katren that despite not always being physically present, has led me doing things that I never imagined. Thank you for being by my side while developing this project and for helping me.

I am grateful for all support from the ProVANT Project members, my friends Gabriel Manoel, Juliano Grigulo, Henrique Misson, Diego Sales, and Rosane Passarini. Thank you for your friendship and support during the writing of this thesis, because without you this work would not be possible.

I would also like to thank my friends, especially Rodrigo Donadel, Martin Bloedorn, Vinicius Stramosk, Gabriel Fernandes, and Richard Andrade, for your support to make this thesis possible, and the text easier to read. Thanks for the friendship started at the Federal University of Santa Catarina (UFSC) and that will be continue for the rest of our lives.

I want to thank my PhD colleagues of automation and systems engineering from UFSC, thanks for the friendship and companionship, as well as for the discussions that have supported the construction of this thesis.

A special thank is devoted to Eduardo Tovar and David Pereira, my internship advisors from Cister in Oporto city Portugal. Thank you for supporting me during this period, especially for the discussions that contributed enormously to my thesis. I want to say thank you to my Portuguese friend Claudio Maia, who worked with me on Cister, thank you for the all that you have done. During the internship I also made some Brazilian friends, especially Marcel Bueno and João Rogano, who I would like to thank for the support, friendship and for making this period far from home easier.

I would like to thank the postgraduate program in automation engineering and systems (PPGEAS) from UFSC, and the Brazilian research agencies CAPES and CNPq for their support and financial contribution to the accomplishment of this work.

I would also like to thank to my advisor professor Leandro Becker, for the opportunity given to me and for supporting me with great disposition throughout my thesis. I appreciate all you have made for me.

Finally, I want to thank all those who have contributed in some way to making this research possible.

Once you have tasted flight, you will  
forever walk the earth with your eyes  
turned skyward, for there you have been,  
and there you will always long to return.  
Leonardo da Vinci



## RESUMO

O projeto de sistemas ciberfísicos (CPS) é considerado uma atividade complexa, sendo composto por diferentes fases, essenciais para sua concepção. Neste sentido, a definição detalhada das fases de projeto se faz necessária, visando facilitar o projeto das aplicações e auxiliar na representação das suas características. Algumas dessas fases têm ampla discussão por parte da comunidade da engenharia, tais como o desenvolvimento dos sistemas de controle, por exemplo. Outras, no entanto, têm um menor grau de detalhamento, ou seja, as atividades que devem ser desenvolvidas não são amplamente discutidas ou detalhadas, dificultando a sua aplicação. Dentre estas temos a especificação dos subsistemas de sensoriamento e atuação, integração de processos de verificação formal, entre outras. Essas etapas, apesar de menos discutidas, também são essenciais no escopo do desenvolvimento dos CPS, pois suportam a especificação e validação de propriedades das aplicações, assim como são responsáveis pela integração da aplicação com o ambiente de atuação. No projeto dos CPS, um número considerável de métodos de desenvolvimento está disponível na literatura, visando guiar os desenvolvedores nas tarefas de modelagem, porém eles não apresentam adequado nível de detalhamento para as atividades supracitadas. Nesse contexto, este trabalho propõe o desenvolvimento de um método integrado que auxilie no processo de modelagem e integração dos CPS, mais especificamente dos Veículos Aéreos Não Tripulados (VANTs). O método proposto busca integrar os processos de modelagem funcional, de arquitetura, integração de sensores e atuadores e verificação formal, contribuindo no projeto do sistema embarcado, bem como na interface com o conjunto de sensores e atuadores, culminando, conseqüentemente, na construção das aplicações. O método proposto é baseado na engenharia dirigida a modelos (MDE) e sua abordagem busca permitir a construção automatizada dos modelos, garantindo a manutenção das características da aplicação durante todo o processo de desenvolvimento, permitindo a integração dos modelos gerados e auxiliando na validação das propriedades do sistema. Aliadas ao método proposto, duas ferramentas foram desenvolvidas, denominadas *ECPSModeling* e *ECPSVerifier*. Estas têm por objetivo dar suporte ao processo de desenvolvimento. O *ECPSModeling* opera na transformação do modelo funcional para o modelo de

arquitetura, permitindo a integração das características de sensores e atuadores. Já o *ECPSVerifier* atua na extração do comportamento do sistema, transformando o modelo de arquitetura em um modelo de comportamento representado por autômatos temporizados, permitindo a aplicação da técnica de verificação formal *model checking*. Visando detalhar o método proposto e as ferramentas desenvolvidas, esses são aplicados ao projeto de um VANT birotor na configuração Tilt-rotor. Dessa forma, objetiva-se municiar o processo de desenvolvimento dos CPS, em especial dos VANTs, descrevendo suas principais fases de desenvolvimento.

**Palavras-chave:** Projeto de CPS, VANT, Engenharia dirigida por modelos, Transformação de Modelos



## RESUMO EXPANDIDO

### Introdução

Os sistemas ciberfísicos (CPSs) são descritos como aplicações caracterizadas pela intensa interação com o ambiente em que estão inseridas. Os CPS são definidos como sistemas complexos, tipicamente aplicados ao controle de dispositivos eletromecânicos. No ambiente dos CPS plataformas embarcadas e monitores em rede são utilizados visando o controle de processos físicos, geralmente com o uso de loops de retroalimentação onde os processos físicos e computacionais afetam um ao outro (LEE; SESHIA, 2015; ALUR, 2015).

O projeto de CPS, especialmente dos veículos aéreos não tripulados (UAV), é considerado um processo gradual composto por um conjunto de etapas que visam detalhar as características da aplicação e validar a informação fornecida por meio de simulações e análises (LEE; SESHIA, 2015; JENSEN; CHANG; LEE, 2011a; BECKER et al., 2010). Devido a esta característica seu processo de desenvolvimento requer uma maior atenção durante a fase de concepção, de forma a gerar um produto que atenda aos requisitos de projeto (MARWEDEL, 2010).

Neste sentido, o projeto de CPS é considerado uma atividade complexa, sendo composta por diferentes fases essenciais para sua concepção. Considerando a complexidade associada ao desenvolvimento destes projetos a definição detalhada de suas fases se faz necessária, auxiliando no projeto e construção das aplicações.

Considerando o processo de desenvolvimento dos UAVs, alguns desafios são observados, descrevendo o desenvolvimento do sistema de controle, a especificação e integração do seu conjunto de dispositivos e a avaliação e validação de suas propriedades. Considerando o projeto do sistema de controle, este é descrito como um processo complexo que exige o desenvolvimento de algoritmos sofisticados. A especificação e integração do conjunto de dispositivos à aplicação CPS também é descrita como uma tarefa não trivial, sendo necessário a avaliação de diferentes características, assim como, a definição da plataforma embarcada para integração destes componentes também é um desafio. O projeto dos UAVs requer um alto grau de confiança quanto a validação de suas propriedades, sendo este processo muitas vezes realizado somente por meio de simulações, atividade esta que não é suficiente para garantir a avaliação e validação da aplicação, sendo necessário o uso de técnicas adicionais associadas ao projeto das

aeronaves.

Além dos desafios descritos, considerando os diferentes métodos de desenvolvimento aplicados a CPS descritos na literatura é observado que algumas destas fases são mais discutidas do que outras. Neste sentido, é verificado que alguns pontos do processo de desenvolvimento não são suficientemente detalhados, gerando dúvidas aos times de projeto. Da mesma forma, apesar de muitas das propostas considerar como base a engenharia dirigida por modelos (MDE), também é observado que não há grande integração entre as diferentes representações de projeto geradas, dificultando a manutenção das informações durante o desenvolvimento do projeto.

## **Objetivo**

Esta tese tem como objetivo prover contribuições para o processo de desenvolvimento de CPS, permitindo a especificação das propriedades dos subsistemas, a integração dos dispositivos e o suporte para avaliação e validação de propriedades por meio do uso de técnicas de verificação formal. As contribuições apresentadas serão aplicadas principalmente ao processo de desenvolvimento de Veículos Aéreos Não tripulados.

## **Metodologia**

A metodologia utilizada para o desenvolvimento desta tese se baseia em três componentes principais: (i) metodológicos, (ii) modelagem do conjunto de sensoriamento e atuação, e (iii) verificação formal.

Considerando (i) uma proposta de método de desenvolvimento aplicada a UAVs foi desenvolvida, buscando sistematizar o processo de desenvolvimento e detalhar tanto as etapas específicas do projeto de UAVs, quanto atividades de gerenciamento de projeto. Desta forma, se busca guiar o processo de projeto da sua concepção à validação pelo cliente. O método proposto é baseado na MDE e prevê a construção de representações complementares para mapeamento das características da aplicação. A construção de alguns destes modelos é automatizada pela aplicação de processos de transformação de modelos, permitindo a geração de novas representações com base em seus dados de entrada.

O Item ii propõe contribuições aplicadas ao processo de modelagem do conjunto de sensoriamento e atuação das aplicações. Neste sentido a extensão de um processo de transformação de modelos foi proposta. Neste processo, modelos funcionais (Simulink) são

utilizados como base para geração de modelos de arquitetura (AADL). Neste sentido, visando ampliar o nível de detalhe dos dispositivos do sistema, assim como permitir a especificação do conjunto de sensores e atuadores durante o processo de transformação, etapa anteriormente não coberta pelo processo de transformação original, foi desenvolvida como extensão do processo de transformação proposto, permitindo aos desenvolvedores a especificação das características de sensores e atuadores, assim como a definição de funções e tarefas responsáveis por prover a interface com os dispositivos.

Visando integrar técnicas de verificação formal ao processo de desenvolvimento de UAV (Item iii), um segundo processo de transformação de modelos foi proposto. Neste características são integradas ao modelo de arquitetura AADL, permitindo extração do comportamento do sistema representado por meio de autômatos temporizados, os quais serão utilizados como base para o processo de avaliação e validação do sistema por meio do uso do *Model Checking* na ferramenta UPPAAL.

## **Resultados e Discussão**

Os resultados apresentados nesta tese de doutorado são divididos em conformidade com os três componentes principais da tese mencionados anteriormente. Sobre o primeiro, o Capítulo 4 desta tese apresenta o desenvolvimento do método aplicado ao projeto de UAVs, bem como a sua validação com o desenvolvimento de um UAV bi-rotor de configuração tilt-rotor aplicado a missões de busca e resgate. Os resultados apresentados mostram que o método proposto busca detalhar não só as atividades técnicas, relacionadas ao projeto do UAV, como também descreve atividades gerenciais de projeto, além disso dois processo de transformação de modelos são associados ao método proposto descrevendo suporte à geração de modelos e contribuindo para o aumento na integração destas representações e a manutenção da informação de projeto durante todas as fases de desenvolvimento.

Referente ao segundo componente, o Capítulo 5 desta tese apresenta a extensão de um processo de transformação pre-existente (de modelo funcional Simulink para modelo de arquitetura AADL). Neste uma etapa intermediária de processamento foi adicionada visando dar suporte a análise e especificação dos subsistemas de sensoriamento e atuação, processo este que era considerado como uma etapa a ser realizada posteriormente na proposta de transformação original. Para dar suporte ao processo de transformação a ferramenta *ECPS Modeling* foi desenvolvida auxiliando na representação e organização

das informações. Os resultados apresentados indicam a viabilidade de inclusão de informações ainda durante o processo de transformação, permitindo a geração de um modelo de arquitetura que integra o sistema de controle ao seu conjunto de sensores e atuadores. Visando a validação do processo desenvolvido a ferramenta foi aplicada ao projeto de um UAV detalhando as etapas aplicadas bem como o modelo de saída gerado.

Considerando a integração do processo de verificação formal (terceiro componente), o Capítulo 6 descreve o desenvolvimento do processo de transformação do modelo de arquitetura AADL em autômatos temporizados. Esses são submetidos ao processo de verificação formal por meio do uso da técnica *Model Checking* na ferramenta UPPAAL. Esse processo tem suporte da ferramenta *ECPS Verifier*, desenvolvida no escopo desta tese para permitir a transformação automatizada dos modelos. Os resultados obtidos demonstram que com o refinamento aplicado ao modelo de arquitetura AADL se faz possível extrair o comportamento da aplicação, permitindo a avaliação e validação das propriedades do sistema. A ferramenta desenvolvida foi aplicada ao processo de desenvolvimento de um UAV para detalhamento e validação das suas propriedades.

## **Considerações Finais**

Neste tese contribuições foram apresentadas visando aprimorar o processo de desenvolvimento dos CPS, em especial dos UAVs. Essas foram propostas após uma extensiva análise de diferentes métodos para projeto de CPS, considerando as características requeridas no projeto de UAVs. Estudos foram realizados para avaliar a integração do conjunto de dispositivos do sistema nestas aplicações. A aplicação de métodos de verificação formal a esse processo também foi avaliada. Com base nos estudos realizados três contribuições foram propostas, com objetivo de prover um maior detalhamento de fases não tão amplamente discutidas no projeto de UAVs, descrevendo o método de projeto aplicado a UAVs e dois processos de transformação de modelos que visam dar suporte ao método proposto.

Em resumo esta tese descreve uma solução integrada aplicada ao projeto de UAVs detalhando suas fases e atividades. Esse método é complementado por meio de dois processos de transformação de modelos que permitem a integração das características dos dispositivos do sistema e a avaliação e validação das propriedades por meio do uso da verificação formal. O método proposto foi aplicado ao projeto de um UAV tilt-rotor UAV. Os resultados obtidos demonstram que

com a aplicação do referido método representações complementares são geradas ao longo do processo de desenvolvimento do projeto, garantindo um maior detalhamento das informações. Por meio do uso dos processos de transformação se busca reduzir o tempo de projeto e contribuir para tornar o processo menos propenso a erros.

**Palavras-chave:** Projeto de CPS, VANT, Engenharia dirigida por modelos, Transformação de Modelos



## ABSTRACT

The design of a Cyber-Physical System (CPS) is defined as a complex activity, being composed of a set of design phases devoted to model application characteristics. In this sense, detailing the phase characteristics is required in order to help the design teams during the project design, and to aim the support of the correct application characteristics representation. However, despite some of these phases, such as the control systems design, being discussed at length by the engineering community, other phases have less detailed studies, i.e., the design activities that compose these phases are not so well documented. In these sense, it is required more experience from the design teams to perform activities such as, the sensing and actuation subsystems representation, and the integration of formal verification methods on the design process, among others. Despite the lack of information related to them, these phases are essential to the CPS design, by the fact that they support application characteristics representation and the properties validation, as well as, they also provide the integration between designed system and their environment. Regarding the CPS design process, different methods are available in the literature, aiming to guide the designers to perform the modeling tasks. However, these approaches do not provide enough information related to those described activities. In this context, this thesis proposes an integrated method applied to CPS design, more specifically devoted to the Unmanned Aerial Vehicles (UAV) design. That proposed method aims to integrate different modeling processes such as functional, architectural, sensor and actuator integrations, and formal verification design processes. Based on the proposed activities this method aims to support the UAV embedded system design, and allow the integration between the embedded platform and the set of system devices. The Model Driven Engineering (MDE) is used as basis to the proposed approach, and aims to support the automated model generation based on the application characteristics. It is intended to ensure the maintainability of the system information over all the design steps and provide the property evaluation and validation, considering the model transformation principles. Two different tools are designed with the proposed method, the *ECPSModeling* and the *ECPSVerifier*, for supporting the design activities. The *ECPSModeling* provides the transformation process from functional model to architectural model,

in order to integrate sensor and actuator characteristics. On the other hand, the *ECPSVerifier* provides the CPS behavior representation, based on the architectural model, by using timed automatas, which allows the formal verification evaluation by performing model checking. The proposed method and the designed tools are applied on the project of a tilt-rotor UAV design. The details of the method proposed in this thesis are demonstrated by performing the UAV project, described as a case study.

**Keywords:** CPS design process, UAV, Model Driven Engineering, Models transformation



## LIST OF FIGURES

Figure 1	Concept of CPS.....	1
Figure 2	UAV architectural representation.....	3
Figure 3	Model Transformation Overview.....	15
Figure 4	AADL Components.....	21
Figure 5	Relations between AADL components.....	22
Figure 6	UAV design method workflow.....	59
Figure 7	Rapid Intervention Vehicle.....	70
Figure 8	VTOL-CP physical model.....	73
Figure 9	VTOL-CP UAV architecture.....	74
Figure 10	VTOL-CP UAV Functional Model.....	74
Figure 11	High-level view of the UAV architecture.....	75
Figure 12	High-view of UAV Architectural model.....	76
Figure 13	Position estimation task behavior.....	78
Figure 14	GPS behavior representation.....	79
Figure 15	Main activities and artifacts of the method to develop CPS.....	84
Figure 16	UAV Simulink model: first hierarchical level.....	87
Figure 17	ECPSModeling process workflow.....	88
Figure 18	ECPSModeling definition of mathematical block.....	89
Figure 19	ECPSModeling analyzing the block inputs.....	90
Figure 20	ECPSModeling pre=processing functions definition.....	91
Figure 21	ECPSModeling actuators definition.....	92
Figure 22	Define the actuation software structure.....	93
Figure 23	ECPSModeling output analysis.....	94
Figure 24	Define post-reading functions.....	95
Figure 25	<i>PositionEst</i> function definition.....	95
Figure 26	ECPSModeling sensor specification.....	96
Figure 27	ECPSModeling sensing threads specification.....	97
Figure 28	UAV model with sensing and actuation process.....	99
Figure 29	AADL representation of sensing and actuation process.....	99
Figure 30	<i>ECPS Verifier</i> Top View.....	103
Figure 31	ECPS Verifier Tasks Behavior and Interferences	

evaluation. ....	104
Figure 32 ECPS Verifier Schedulability Analysis. ....	107
Figure 33 AADL meta-model. ....	111
Figure 34 UPPAAL meta-model. ....	112
Figure 35 Scheduler model. ....	114
Figure 36 UAV Behavior model. ....	115
Figure 37 AADL design conventions. ....	117
Figure 38 UAV model with sensing and actuation process. ....	118
Figure 39 AADL representation of sensing and actuation process. ....	119
Figure 40 Split of UAV model with a subset of devices. ....	120
Figure 41 AADL representation of position estimation components. ....	120
Figure 42 GPS fault tree representation. ....	121
Figure 43 AADL GPS error representation. ....	121
Figure 44 AADL position thread behavior. ....	122
Figure 45 Position estimation task. ....	122
Figure 46 GPS template. ....	123
Figure 47 AADL position thread behavior. ....	124
Figure 48 Tasks template. ....	125
Figure 49 Evaluated UAV properties. ....	127
Figure 50 Obtained results of evaluated UAV properties. ....	128

## LIST OF TABLES

Table 1	Evaluation of CPS design works.....	36
Table 2	Evaluation of Integration of Sensors and Actuators .....	44
Table 3	Evaluation of CPS Formal Verification.....	50
Table 4	UAV control stability requisite.....	72
Table 5	UAV load transportation requisite.....	73



## LIST OF ACRONYMS

CPS	Cyber-Physical System
IT	Information Technology
UAV	Unmanned Aerial Vehicle
MDE	Model-Driven Engineering
CASE	<i>Computer-Aided Software Engineering</i>
AADL	<i>Architecture Analysis and Design Language</i>
AST	Assisted Models Transformation
DSML	Domain-Specific Modeling Languages
QoS	Quality of Service
M2M	Model to Model
M2C	Model to Code
OMG	Object Management Group
MDA	Model Driven Architecture
PIM	Platform-Independent Models
PSM	Platform-Specific Model
EMF	Eclipse Modeling Framework
XML	eXtensible Markup Language
UML	Unified Modeling Language
MoC	Model of Computation
SAE	Society of Automotive Engineers
SELT	State/Event LTL model-checker
TTS	Timed Transition Systems
RTOS	Real-Time Operating System
PM	Platform Model
CERBERO	Cross-layer model-based framework for multi-objective design of reconfigurable systems in uncertain hybrid environments
MDD	Model-Driven Design
FR	Functional Requirements
NFR	Non-Functional Requirements
VTA	Virtual Target Architecture
TSAM	Timed Abstract State Machines

TPN	Timed Petri Nets
NPTA	Networks of Priced Timed Automata
VTOL-CP	Vertical Take-Off and Landing Convertible Plane
UFSC	Federal University of Santa Catarina
UFMG	Federal University of Minas Gerais
SAR	Search and Rescue
IMU	Inertial Measurement Unit
GPS	Global Position System
ESC	Electronic Speed Controller
EA	AADL Error Annex
BA	AADL Behavior Annex

# CONTENTS

<b>1</b>	<b>INTRODUCTION</b> .....	1
1.1	MOTIVATION.....	5
1.2	OBJECTIVES .....	7
1.3	OUTLINE.....	8
1.4	LIST OF PUBLICATIONS .....	9
<b>2</b>	<b>CONCEPTS, TECHNOLOGIES AND TECHNIQUES</b> .....	11
2.1	MODEL-DRIVEN ENGINEERING .....	11
2.1.1	Models and Metamodels .....	13
2.1.2	Model Transformation .....	15
2.2	TOOLS AND LANGUAGES APPLIED TO CPS DESIGN	16
2.2.1	MATLAB/Simulink .....	18
2.2.2	Architectural Analysis Design Language - AADL..	20
2.2.3	OSATE 2 .....	23
2.2.4	Formal Verification.....	24
2.2.5	Model Checking.....	26
2.2.6	UPPAAL .....	28
2.3	SUMMARY .....	30
<b>3</b>	<b>STATE OF THE ART</b> .....	31
3.1	RELATED WORKS EVALUATION CRITERIA .....	31
3.1.1	Criteria for CPS Design Methods .....	32
3.1.2	Criteria for Integration of Sensors and Actuators .	32
3.1.3	Criteria for Formal Verification .....	33
3.2	CYBER-PHYSICAL SYSTEMS DESIGN METHODS ...	34
3.2.1	Overview .....	34
3.2.2	Evaluation and Discussion .....	36
3.3	INTEGRATION OF SENSORS AND ACTUATORS ...	42
3.3.1	Overview .....	42
3.3.2	Evaluation and Discussion .....	43
3.4	FORMAL VERIFICATION ON CPS DESIGN.....	48
3.4.1	Overview .....	48
3.4.2	Evaluation and Discussion .....	49
3.5	SUMMARY AND ADDITIONAL REMARKS .....	54
<b>4</b>	<b>DESIGN METHOD FOR UNMANNED AERIAL VEHICLES</b> .....	57
4.1	PROPOSED APPROACH .....	58
4.1.1	Design Activities.....	58

4.2	DESIGN OF A VTOL-CP UAV.....	69
4.2.1	UAV Method Applied to the project Design .....	70
4.3	SUMMARY .....	80
<b>5</b>	<b>SENSING AND ACTUATION SUBSYSTEMS DESIGN .....</b>	<b>83</b>
5.1	RESEARCH CONTEXTUALIZATION .....	84
5.2	PROPOSED APPROACH AND RELATED DESIGN ACTIVITIES .....	85
5.2.1	Case Study .....	86
5.2.2	Design Activities.....	87
5.2.3	Output Model Generated by the Tool.....	98
5.3	SUMMARY .....	99
<b>6</b>	<b>INTEGRATING FORMAL VERIFICATION INTO THE UAV DESIGN .....</b>	<b>101</b>
6.1	FORMAL VERIFICATION OF AADL ARCHITECTURAL MODELS.....	102
6.1.1	Phase 1: Evaluation of Tasks and Interferences....	104
6.1.2	Phase 2: Schedulability Analysis .....	106
6.1.3	Final Remarks .....	108
6.2	MODEL TRANSFORMATION TOOL <i>ECPS VERIFIER</i> .....	110
6.2.1	Related Metamodels .....	110
6.2.2	Transformation Process .....	110
6.2.3	Auxiliary Components .....	113
6.3	DESIGN OF SENSING AND ACTUATION SUBSYSTEMS OF AN UAV .....	118
6.3.1	Evaluation of tasks behavior and interferences ....	119
6.3.2	Schedulability Analysis.....	123
6.4	UAV PROPERTIES EVALUATION .....	125
6.5	SUMMARY .....	128
<b>7</b>	<b>CONCLUSIONS .....</b>	<b>131</b>
7.1	FUTURE WORKS .....	133
	<b>Bibliography .....</b>	<b>135</b>

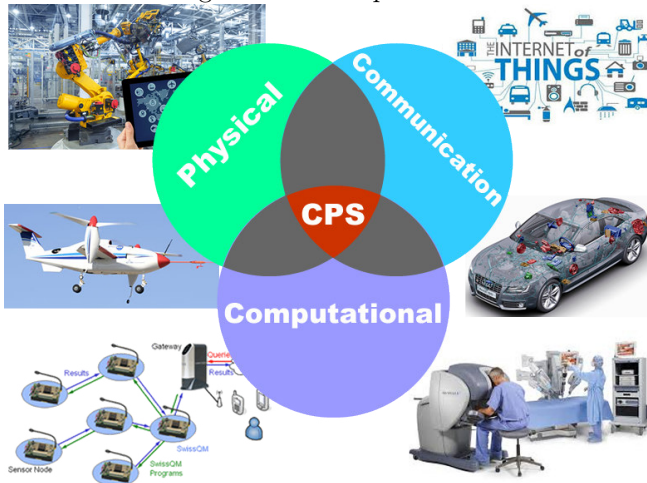


## 1 INTRODUCTION

Cyber-Physical Systems (CPSs) are applications characterized for performing intensive interaction with the surrounding environment. CPSs consist of complex systems typically applied to control electro-mechanical devices. In the CPS environment embedded computers and network monitors are applied to control the physical processes, usually with feedback loops where physical and computational processes affect each other. (LEE; SESHIA, 2015; ALUR, 2015).

Analyzing the CPS structure at least three main components are observed in this sketch (Fig. 1). The first describes the physical plant, defining the CPS “physical” part. The physical plant represent the components that are not executed by computers or digital networks, and it can include mechanical parts, biological or chemical processes, or human operators. The second part relates to computational platforms, which consist in a set of devices coupled with computers, and one or more operating systems. The third defines the communication interfaces, which provides the mechanisms for information exchange. In this sense, the platforms and the network interface provide the “cyber” part of the cyber-physical system(ALUR, 2015).

Figure 1 – Concept of CPS.



Regarding the 20th century Information Technology(IT) revolution, the CPS popularization was increased by the technological evolution, that provided components with higher processing power and more energy efficiency. In the same way, the communication protocols and the network infrastructure has evolved, allowing more interaction and information exchange. However, despite this evolution the CPS design requires understanding the joint dynamics of computers, software, networks, and physical processes. In this sense, this design process is considered a multidisciplinary task, which involves different teams working in a collaborative way to properly address the application features (DERLER; LEE; VINCENTELLI, 2012).

Over the last years the CPS has been applied to different environments, requiring particular levels of reliability and safety according the problem domains. These domains include robotic manufacturing systems; electric power generation and distribution; process control in chemical factories; distributed computer games; transport of manufactured products; heating, cooling, and lighting into smart buildings; people movers such as elevators; bridges that monitor their own state of health; the automotive industry; and aerospace applications (LEE, 2008).

Regarding the aerospace environment, different applications have been designed over the CPS scope such as satellites, spacecrafts, and Unmanned Aerial Vehicles (UAVs). In relation to UAVs its observed that the use of these aircrafts has grown tremendously in recent years, mainly due to technological innovation in fields like control design, estimators, and system components (PAPACHRISTOS et al., 2011).

Initially, the UAVs were widely used in military applications, due to their flexibility to integrate into different environments and their ability to be remotely operated and transmit information in real-time, being applied into surveillance, and reconnaissance mission for example (KEANE; CARR, 2013). However, the UAVs also began to be used in civilian applications, promoting much research. These vehicles have shown potential for missions such as remote sensing, cargo transportation, search and rescue, precision agriculture, border monitoring, among others (COSTA et al., 2012; NAIDOO; STOPFORTH; BRIGHT, 2011; PING et al., 2012).

Evaluating the different UAV characteristics and configurations, it is possible to identify two big UAV groups of architectures in the literature: fixed wing and rotary wing. The fixed wing aircrafts are characterized by high autonomy and high speed (Fig. 2a); on the other

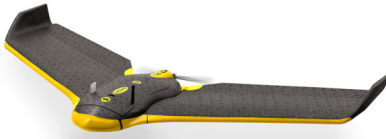
hand, rotary wings (namely helicopters), have as main characteristic the good maneuverability (Fig. 2b).

Despite these defined groups another UAV category has emerged, describing the Tilt-rotor aircraft, which is an aerial vehicle whose design is between both these two architectures, propelled by two tiltable rotors. One of the most notable aircraft is the Bell-Boeing V-22 Osprey, which is used by US military to perform several kinds of missions such as troops or military equipments transportation, as shown in Fig. 2c. Nowadays other Tilt-rotor UAVs has been developed such as, TR918 Eagle Eye shown in Fig. 2d, whose construction began in 1993 with its final version being released in 1998. This was designed and built for Bell by the research company Scaled Composites.

Figure 2 – UAV architectural representation.

(a) Ebee - fixed wing UAV.

(b) TURAC - rotatory wings UAV.



Source: SenseFly (2018).

Source: Cai et al. (2008).

(c) Bell-Boeing V-22 Osprey.

(d) Bell Eagle Eye TiltRotor UAV.



Source: UAVGLOBAL (2008).

Regarding the UAV design process, considerable challenges are observed to provide such system work. First, controlling the vehicle is not trivial task and sophisticated control algorithms are required.

Secondly, specifying and integrating the set of required devices into the CPS application is not a trivial task, and several characteristics need to be evaluated, as well as the definition of the embedded platform to integrate these components is also a challenge. Thirdly, the vehicle needs to operate in a context, interacting with its environment. It might, for example, be under the continuous control of a watchful human who operates it by remote control. Or it might be expected to operate autonomously, to take off, performing a mission then returning and landing.

Providing the autonomous operation is enormously complex and challenging, because it cannot benefit from the watchful human. The autonomous operation demands more sophisticated sensors, the vehicle needs to keep track of where it is, requires that the aircraft senses the obstacles, and it needs to know where the ground is. These vehicles also needs to continuously monitor their own health, in order to detect malfunctions and react to them so as to contain the damage. It requires detailed modeling of the environment dynamics, and a clear understanding of the interaction between these dynamics and the embedded system.

In this context, different CPS methods are proposed nowadays, aiming to provide a guideline to the design teams to build these applications. Some of these methods are based on Model-Driven Engineering (MDE) (SCHMIDT, 2006) in order to support the capture and representation of CPS characteristics. By performing MDE principles, complementary models can be created and different system dimensions represented (BECKER et al., 2010; JENSEN; CHANG; LEE, 2011b; DERLER; LEE; VINCENTELLI, 2012).

An MDE characteristic propose the use of model transformation aiming to automate the design tasks. Theses processes are applied to provide the automated models generation, where based on a source model, a target complementary model can be generated, as well as the application code derived.

Despite the different CPS methods proposed, and considering the UAV design complexity, at least three difficulties are observed on theses processes. The first describes that some design steps are more discussed than others, such as the control systems design, being discussed at length by the engineering community, while sensing and actuation subsystems design less detailed studies.

The second problem is related to the weak integration between the project phases and the tools support. This problem is caused by the difficulty of providing the model mapping, making it difficult

to maintain any information during the design process. Similarly, tools and methods sometimes do not provide enough characteristic representation for the UAV design process needs, requiring that manual work be performed by the development team, which can be subject to project errors.

Integration of formal verification methods applied to validate the UAV project properties is observed as a third difficulty. Some methodologies do not integrate these techniques into their approaches. In this way, the validation process of application properties becomes difficult and not precise, often this validation is based on project team experience.

## 1.1 MOTIVATION

Design CPS applications, specially UAVs, is considered a gradual process composed by a set of steps that detail the application characteristics and validate the provided information by performing simulations and analysis (LEE; SESHIA, 2015; JENSEN; CHANG; LEE, 2011a; BECKER et al., 2010). Due to this process complexity, the application design requires a higher carefully during its conception, in order to provide a product that fulfill its requirements (MARWEDEL, 2010).

Computer-Aided Software Engineering (CASE) tools based on functional characteristics and that support simulations are typically used to represent the CPS behavior (functional representation) (LEE; SESHIA, 2015) such as the Ptolemy (BERKELEY, 1999) and the Simulink (MATHWORKS, 2018). However, beside they provide support to generate application code, they do not present enough support to represent architectural aspects (GONCALVES et al., 2013b).

Represent the architectural characteristics of CPS application is required independently of the applied approach. These aspects usually are described by creating an architectural model, describing the integration between software and hardware components.

To provide the architectural properties representation languages are applied to properly describe these characteristics. An example of these languages is defined as Architecture Analysis and Design Language (AADL) (FEILER; GLUCH; HUDAK, 2006), that have been extensively applied on critical embedded systems design (ZHAO; MA, 2010). Based on the language properties, application characteristics can be evaluated, such as temporal characteristics (scheduling and flow

latency), and physical properties (weight, power consumption) (FEILER; GLUCH, 2012).

Perform formal verification methods to evaluate and validate the system properties is cited by different authors as essential to ensure the CPS properties validation (CORREA et al., 2010; MOON, 1994). In this sense, regarding the UAV complexity this technique is described as essential to ensure that the designed application fulfills its restrictions and can perform the missions. The properties validation is performed confronting the system description and its specifications, validating the system properties (ONEM; GURDAG; CAGLAYAN, 2008). Synchronous languages can be applied to evaluate these properties. Based on these representations characteristics such as liveness, invariance, and reachability can be evaluated (INRIA, 2012).

Another point observed in different approaches regards the fact that despite built different models to represent the application properties, these representations do not have much integration between themselves, i.e. most of these models are manually built. This fact requires that designers put additional effort into representing the application characteristics either for the transition between models or to provide the application code.

This additional effort applied to the design process, usually performed by the design teams in manual tasks and does not ensure the models mapping, by the fact that these characteristics are only based on the design team experience. In addition, performing manual tasks makes it difficult to provide project maintenance, as well as making the project more prone to errors.

Based on the presented information some challenges are verified in the CPS project, for example the integration between the generated models, definition of the set of sensor and actuator and representation of its properties, the integration of the system devices with the embedded platform, and the integration of the formal verification methods during the design phase. These topics are seen as essential to provide the integration between the designed application and the real world. However, a desired integration level between the design steps is not observed, furthermore some activities are less discussed than others.

Regarding the models integration and aim to automate the generation of these representations, techniques that perform model transformation processes can be applied to support the generation of the application models. The Assisted Models Transformation (AST) (PASSARINI, 2014) can be cited as one transformation technique that aims to generate the architectural software representation based on the

functional model.

The AST main objective regards the translation of the control subsystem to the architectural model, leaving other features that are also essential to the application such as, the sensing and actuation subsystems. The AST considers that the integration of these subsystems is an external activity of the transformation process, and its designers responsibility to perform it.

Due to the fact that the CPS design process is composed of a set of phases and that these steps are not properly integrated, performs the CPS design is considered as a complex activity, as well as, maintain these applications is not an easy task. In this sense it is observed that the existing approaches are lacking and do not properly guide the design team. Similarly, the design processes do not properly provide a means for integration of generated models, validation of systems properties or integration of the set of system devices. This characteristics indicate that automating some of these design steps can contribute to the CPS design process especially in relation to UAV design.

## 1.2 OBJECTIVES

The main objective of this PhD Thesis is to provide contributions to the CPS design process, allowing the subsystems specification, integration of system devices, and support for property evaluation by using formal verification. These contributions are especially applied for design of Unmanned Aerial Vehicles.

In order to achieve the main objective, some specific objectives are defined as follows:

1. Study the existent CPS design methods identifying their deficiencies, especially related to UAV design.
2. Propose a design method applied to UAV systems design, providing a teams guideline and systematizing the process activities.
3. Explore the system devices integration in the CPS design process.
4. Provide contributions to integrate device's characteristics in the design process.
5. Investigate the formal verification methods integration on the CPS design process, providing the system properties evaluation and validation.

6. Design tools to support models transformation processes, contributing to device's characteristics integration and to perform the systems formal evaluation.

### 1.3 OUTLINE

In this introductory chapter, the problem of design CPS, especially UAVs, supported by the tools usage to automate some activities was introduced and motivated. The next chapters are organized as follows:

- **Chapter 2** presents the concepts and techniques related to the topic that compose this dissertation such as MDE, model transformation, modeling languages, among others.
- **Chapter 3** provides a survey of the CPS design state of the art, analyzing several related works. Firstly, different CPS design methods are presented. Secondly, a set of approaches that propose the integration of sensors and actuators in the CPS design process are detailed. Thirdly, works that describe the integration of formal verification methods are presented. Finally, a discussion related to these topics is presented.
- **Chapter 4** shows the proposed CPS design method, devoted to built UAV application. The proposed approach describe a set of activities aiming to guide the design teams to the applications construction. A Tilt-rotor UAV design process is presented to steps taken throughout the proposed method.
- **Chapter 5** introduces the integration of the sensor and actuator properties on the CPS design process. This process is supported by the *ECPS Modeling* tool, that performs the transformation from Simulink model to AADL model, integrating the components characteristics. A case study was presented to detail the transformation process.
- **Chapter 6** presents the integration of formal verification methods in the CPS design process. In this way, a transformation process is performed from AADL models to UPPAAL timed automata, in order to support the system properties evaluation. This approach is supported by a tool named *ECPS Verifier*, and detailed evaluating a Tilt-rotor UAV.



- **Chapter 7** summarizes the contributions and results presented in this thesis and suggests possible future research lines.

#### 1.4 LIST OF PUBLICATIONS

The following papers were published along this PhD:

- GONÇALVES, F. S.; BECKER, L. B. Preparing cyber-physical systems functional models for implementation. In: **V Brazilian Symposium on Computing System Engineering (SBESC 2015)**. Foz do Iguaçu - PR: [s.n.], 2015.
- GONÇALVES, F. S. et al. Vant autônomo capaz de comunicar com uma rede de sensores sem fio. In: **X Congresso Brasileiro de Agroinformática (SBIAGRO 2015)**. Ponta Grossa - PR: [s.n.], 2015.
- GONÇALVES, F. S.; BECKER, L. B.; RAFFO, G. V. Managing CPS complexity: Design method for Unmanned Aerial Vehicles. In: **2016 1st IFAC Conference on Cyber-Physical & Human-Systems**. [S.l.: s.n.], 2016.
- GONÇALVES, F. S.; BECKER, L. B. Model driven engineering approach to design sensing and actuation subsystems. In: **2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.: s.n.], 2016. p. 1–8.
- GONÇALVES, F. S. et al. Formal verification of aadl models using uppaal. In: **VII Brazilian Symposium on Computing Systems Engineering (SBESC 2017)**. Curitiba - PR: [s.n.], 2017.



## 2 CONCEPTS, TECHNOLOGIES AND TECHNIQUES

Represent the CPS application characteristics in order to correctly describe its properties is defined as one of the CPS design challenges, especially when the UAV systems design is considered. This complexity is inherent of this applications environment, as well as on the required guarantees that can be applied to these systems (LEE; SESHIA, 2015). This process is composed of a set of phases, aiming to detail the application characteristics. In this sense, aiming to support the design activities, different tools and techniques may be applied (JENSEN; CHANG; LEE, 2011b; BECKER et al., 2010).

Aiming to improve the model characteristics and make the designers work less prone to errors, different technologies may be applied into the UAV design. In this context, techniques, modeling languages, and tools are applied to these design processes are presented in this chapter.

### 2.1 MODEL-DRIVEN ENGINEERING

MDE (SCHMIDT, 2006) the method applied to capture and represent systems characteristics. By the MDE usage complementary models may be created, representing different system viewpoints such as, the physical representation, the control system, the architectural structure, the functional representation, the electrical components, among others.

This technique is focused on representing the application characteristics by model construction. Different to the traditional approaches, MDE is not centered on coding but instead defining the specification of the system characteristics before writing the application code (ALANEN et al., 2004; SCHMIDT, 2006). MDE was proposed by Stuart Kent (KENT, 2002) as an approach that describes the application design, analysis, and representation.

The design of complementary models applied to represent the system characteristics is proposed by MDE, and based on this characteristic different viewpoints are created. In addition to the model design, the MDE support the automated code generation process, where based on the designed models, the software structure, to be applied to the target platform, can be extracted (ALANEN et al., 2004).

During the evolutionary CPS design process, different models

are generated, providing more project information, and increasing the team's productivity. However, these structures need to be integrated in order to compose the CPS application. By the use of MDE standards, the model representation should be reused, simplifying the design process. The MDE aims to face the platform's complexity as well as the inability of the third generation languages to effectively express the domain application concepts. The MDE introduce technologies that incorporate (SCHMIDT, 2006):

- ***Domain-Specific Modeling Languages (DSML)***: Languages that support the definition of an application's structure, system behavior, and domain characteristics. By using these languages, properties are defined based on meta-models, that represent the relation between domain concepts, which furthermore express the model semantic associated with its restrictions;
- **Transformation engines and generators**: That analyze the model aspects and synthesize different artifacts, such as source code, alternative model representations, among others. The synthesis of these artifacts ensures consistency between application implementations and information analysis associated with functional analysis and Quality of Service (QoS) requirements captured in the models.

This technique proposes model transformation processes, where based on an initial domain application model (defined as source-model), it is possible to generate different representations (defined as target-models), which should be in accordance with the source-model. The model transformation can be carried out in two ways, to generate new representations (Model to Model - M2M), or to provide application code (Model to Code - M2C).

An approach based on MDE was proposed by the Object Management Group (OMG) defining a standard to the platform independent applications design. This standard is defined as Model Driven Architecture (MDA), that is focused on the specification of system functionalities as an independent process detached from the specific platform implementation (OMG, 2003).

Methodologies based on MDA provide the application design by automated mapping processes, attaching the models to its implementations. By MDA usage the OMG aims to provide means by which designers can focus on application requirement definition without regard for implementation aspects of the target platform.

The MDA is based on a mapping concept, where the defined Platform-Independent Models (PIM) are mapped into the Platform-Specific Models (PSM) by the use of automated or semiautomated mechanisms to perform model transformation. The definition of PIM and PSM models is performed with the use of metamodels that describe the syntax, and operational semantic use of each modeling language(LOPES et al., 2006a).

Another MDE approach its defined as Eclipse Modeling Framework (EMF) that provides a framework to design and perform the application code generation based on structured data models (STEINBERG; BUDINSKY; MERKS, 2009). By the use of EMF its possible to build models and perform the java code generation, allowing the design, edition and storage of a designed model instance.

The EMF unify three technologies Java, XML<sup>1</sup> and UML<sup>2</sup> where independent of the design language the provided representation can be considered as a common model with properties of these three languages. These characteristics describe that defining an EMF transformation method this method should be applied to other technologies. The EMF framework has a metamodel that details the model's characteristics, i.e., defines the structure to store the designed representations (ECLIPSE, 2004). The MDA and EMF are considered representative approaches of MDE, describing important concepts such as models, metamodels, and transformation processes (LOPES et al., 2006b).

### 2.1.1 Models and Metamodels

A model is described as a set of formal elements that represent a particular object of study, designed for a specific purpose, which can be evaluated by performing some kind of analysis (MELLOR et al., 2004; MELLOR; CLARK; FUTAGAMI, 2003). Models can also be defined as a real life simplified view (SELIC, 2003), or as a set of prepositions that describe characteristics of the studied objects (SEIDEWITZ, 2003). This representation is essential to the engineering being applied to represent complex problems to be solved, reducing the implementation time and cost of the complex solutions projects (SELIC, 2003).

The MDE makes use of programing and modeling languages, and based on these components models are generated during different

---

<sup>1</sup>eXtensible Markup Language

<sup>2</sup>Unified Modeling Language

project phases, providing not only the system documentation, but being part of its design. By this fact, models are considered as the main artifacts generated during the application project. In this sense, greater accuracy is required for the design of these representations, specifying the system characteristics, and considering them as part of the development process, i.e., these objects are used as a basis for the final solution, by performing transformation processes (MAIA, 2006).

The models can be created as descriptive or prescriptive representations (KÜHNE, 2006; HENDERSON-SELLERS, 2012). Descriptive models detail characteristics of an existing object, while prescriptive models (also named as specification models) represent the object to be implemented. The use of descriptive or prescriptive models is defined according the application context (SEIDEWITZ, 2003; HENDERSON-SELLERS, 2012). In software engineering, models usually represent pieces of the application domain while also defining software characteristics applied to this domain (HESSE, 2006).

A model is considered suitable only if evaluated in a simulation environment that behaves similarly to a real object. Due to the abstraction property (MDE characteristic), the models represent a set of relevant information related to the studied object, i.e., that describe the application subset aspects allowing for properties evaluation. In this sense, aiming to obtain a complete application view, the design process includes the use of multiple complementary models, describing the application characteristics (MELLOR; CLARK; FUTAGAMI, 2003).

Metamodels are created to provide model specification, these representations detail language expresiveness, i.e., specify the set of prepositions to guide the valid model construction (SEIDEWITZ, 2003; GAAEVIC et al., 2006). A model definition is based on its metamodel, that represents a metamodel instance. The model is considered in conformity with its metamodel only if it is syntactically correct, and meets the constraints imposed by the metamodel (KÜHNE, 2006).

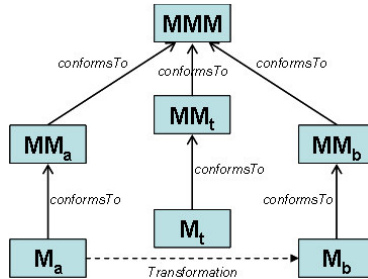
As described by Mellor et al. (2004) a metamodel details the structure, the semantic, and the restrictions of model families, i.e., model groups that share syntaxes and semantics. In this way, a metamodel describes a precise definition and the required rules to create semantic models, and its structure defines the relations between the model's elements.

## 2.1.2 Model Transformation

The model transformation represent the automated model's generation, where based on source model a target representation is provided. This activity describes an important role of the MDE scope (KLEPPE; WARMER; BAST, 2003). The OMG defines the model transformation as a translation process between models in the same system (OMG, 2003). Baudry et al. (2006) on the other hand, describe the model transformation as a kind of language relation.

The definition of a set of rules is required for the transformation process. These structures detail how the source model is translated into an equivalent representation in the target language. The definition of transformation rules defines the way that one or more structures on the source language are represented in the target representation (KLEPPE; WARMER; BAST, 2003). This process is exemplified in the Fig. 3, that represents a top view description of this process, and detail its concepts and relationships.

Figure 3 – Model Transformation Overview



Source: Eclipse (2006).

The transformation process is composed of a source model  $M_a$ , which must comply with its metamodel  $MM_a$ , that will be transformed into a target model  $M_b$ , and the target model must comply with its metamodel  $MM_b$ . The transformation is defined by the use of a transformation model  $M_t$ , which must comply with its transformation metamodel  $MM_t$ . The source and target metamodels ( $MM_a$  and  $MM_b$ ), joined with the transformation metamodel ( $MM_t$ ) shall be in conformity with an metametamodel  $MMM$  (ECLIPSE, 2006).

The transformation is composed of a set of mapping rules to guide the process. By rules usage, the source model components

are related or mapped with the target model components. These rules define mapping standards, specifying relations between different metamodel elements. The identification and characterization of these relations between elements is defined as a mapping scheme (RAHM; BERNSTEIN, 2001).

Lopes et al. (2006a) describes that the mapping provides relations between source and target models, where the target model elements represent the same structure and semantic as the source model. This mapping establishes different relations between elements, and defines one to one relations, one to multiple, and multiple to one.

One-to-one mappings (1:1) represent that one element from the source model is directly mapped to one element from the target model, and they have the same semantic. On many-to-one mappings (n:1) a set of source elements represent the same semantic of one element to the target model, i.e, a target element represents the characteristics and semantic of a set of source elements. Finally the one-to-many mappings (1:n) describe the representation of one element from the source model for a set of elements from the target model, i.e a set of target elements are required to represent the same semantic of an element from the source model.

Based on the MDE principles and aiming to support the CPS design process, different languages and tools have been proposed over the last year. These structures usually provide a design environment and a set of components, providing means by which to detail the application characteristics.

## 2.2 TOOLS AND LANGUAGES APPLIED TO CPS DESIGN

As described in the introduction (Chapter 1) the CPS applications are composed of a set of complementary models, that detail the application characteristics. Regarding the generated models produced during the design process, at least three different representations are provided, defining the functional characteristics (Functional design), the architectural aspects (architectural model), and the system formal evaluation of its set of properties (Formal verification).

Regarding the heterogeneity applied to these models, usually it's difficult to represent all the system characteristics using just one language or tool. In this way, different model languages and tools can be applied to represent the application characteristics. The design process



is usually started by performing the functional modeling, as described by Lee & Seshia (2015). The authors define these representational designs in order to specify the system dynamics. In this way, a set of mathematical expressions are created to represent the application behavior. Based on this behavioral description, control approaches may be proposed, guiding the system according to the imposed requisites.

Jensen, Chang & Lee (2011a) define functional modeling as the process to represent the physical characteristics to be controlled, i.e. models that specify the real system properties by using mathematical expressions. By using this representation control algorithms are defined, and hardware components can be specified, as well as making the evaluation of the designed subsystems a reality.

On the other hand, Alur (2015) states that the functional model is composed by an architecture that aims to support the control system design, being composed of a control system and the physical plant. The control system sends its references in this environment, and the physical plant representation returns the set of system states according to the received inputs.

In this context, aiming to properly represent the application characteristics, different tools should be applied to the functional modeling such as Ptolemy (BERKELEY, 1999), VisualSim (MIRABILIS, 2018), MATLAB/Simulink (MATHWORKS, 1994), among others. These tools provide a set of components that support the representation of the system properties.

The *Ptolemy* project aims to provide an environment aimed at design and evaluation of concurrent systems, real-time systems, and embedded systems. By using an open framework the authors propose an actor oriented approach, that has been in design since 1996. The actors are defined as software components that runs concurrently exchanging information by using communication ports (PTOLEMAEUS, 2014).

The proposed environment provides a hierarchical system construction, where the actors are interconnected by the use of communication ports. These structures are managed by a director component that is responsible for implementation of the model of computation (MoC) . The MoC details how the application runs, supporting the behavioral representation (BERKELEY, 1999).

Different MoC are supported on *Ptolemy* such as discrete events, data flows, reactive synchronous, continuous time model, among others. By using the hierarchical approach, each level can have a director, and different MoC can be applied to its directors. In this way, the

framework provides a means for combining different MoCs in the same project, by using different directors. By using this structure hybrid systems can be designed, providing a means to simulate and evaluate these systems.

The VisualSim is proposed as a modeling and simulation software applied to perform the systems engineering exploration of performance, power and functionality. By the use of a proposed environment, users can construct debug, simulate, analyze their specifications. The systems are built by using a graphical interface that creates the design process with a set of blocks (MIRABILIS, 2018).

The pre-defined blocks have been optimized for simulation performance, and pre-compiled to reduce development time. Custom components can also be created by combining blocks with scripts written in VisualSim Script language.

By using VisualSim interface the user can evaluate logic flows, check the operation correctness, debug, validate requirements and optimize the system to meet the requirements. The designed models can combine different abstraction levels, and different models of computation. VisualSim contains three different simulators describing the timed computation, the untimed digital, and the continuous time.

Another tool widely applied to functional modeling is the MATLAB/Simulink, that provides an environment to design, simulate, and validate the CPS systems. Considering that this tool was applied to this thesis scope, a section was created to represent this tool's characteristics.

### 2.2.1 MATLAB/Simulink

MATLAB is a high-performance design and evaluation tool, and its resources enable the representation of different system characteristics. Designed by *MATrix LABoratory* at the end of 1970, this tool has been widely applied to represent applications characteristics by using mathematical expressions (MATHWORKS, 1994).

A set of extensions are provided by the tool, these components enable different functionalities representation, and the integration with external tools. This set of extensions include the Simulink tool, where the designer can perform the system specification based on a block diagram structure (MATHWORKS, 2018).

By using MATLAB/Simulink different systems can be

represented such as: linear and nonlinear, continuous time, discrete time, and multivariable systems. Its design environment provides a means to specify the system's properties, and simulate these systems, which are usually composed by heterogeneous representations (MATHWORKS, 1994).

The Simulink structure is composed of a set of blocks responsible for implementation of different functions application model construction. Its graphical interface supports the design process and allows the hierarchical applications to be built, defining abstraction levels and making block connection easier (MATHWORKS, 2018).

The implemented Simulink blocks are considered as black boxes, i.e., the designers can only change the set of configuration parameters, but not its functionality itself. However, the tool provides the user with blocks, where the designer can write their own code and integrate it with the existing blocks. A set of inputs and outputs are attached to each Simulink component, that coupled with a set of internal states defines the relation between received inputs and produced outputs.

Regarding the Simulink hierarchical approach applied to the applications design, different abstraction levels can be added. In this way, the complex system can be specified in a set of subsystems, increasing the application details. By the use of Simulink block it is also possible to perform the interface between the designed model and external tools, allowing for example the design of simulation environments.

Besides the functional modeling the CPS design process includes the representation of the architectural characteristics, by using an architectural model. This representation defines aspects that provide the integration between software and hardware elements.

As described by Feiler, Gluch & Hudak (2006) the architectural modeling process is defined as the structural software representation applied to support the system execution, as well as this it allows the integration between hardware and software components. By the use of this representation the required execution characteristics are defined, and a set of system properties evaluation allowed.

Based on the architectural model, system properties can be evaluated ensuring characteristics such as scheduling, latency, weight, power consumption, among others. These analysis allows the designers to evaluate if the defined architecture is capable of supporting the designed application. By the use of this model, software characteristics are defined, specifying properties such as support of control execution and device interface (ZHAO; MA, 2010).

Regarding the design of architectural models different languages can be applied to detail the application aspects. In this sense, AADL has been widely used to design these representations.

### 2.2.2 Architectural Analysis Design Language - AADL

The Architectural Analysis Design Language was designed by The Society of Automotive Engineers (SAE) in 2004 (SAE, 2015). This language was proposed with the aim of being a standard devoted to modeling and designing avionic, aerospace, automotive and robotics applications (FEILER; GLUCH; HUDAK, 2006).

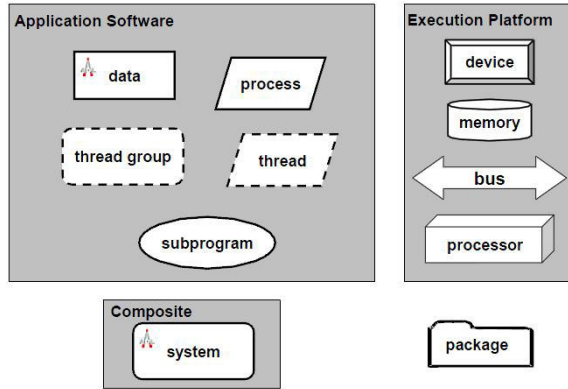
The AADL notation is based on a set of components that describe the system characteristics. By usage of modeling tools the designers can create, analyze, validate real time applications, and perform the code generation for the embedded platforms. The architectural models integrate hardware and software components, describing their characteristics and connections (WANG et al., 2009).

The designers may choose to perform the system representations by using textual, graphical, or XML files. These inputs are supported by the AADL and provide a means by which to express the system properties, helping integration with other tools. The AADL standard provides a hierarchical structure, organized by package usage. These packages provide a means for system specification to be composed of hardware elements (devices, buses, platforms, among others), and software structures (processes, tasks, function calls). By the use of software and hardware structures the system properties are detailed, and the component's integration is represented (ZHAO; MA, 2010).

The set of resources covered by the AADL are described in Fig. 4, being organized essentially into three categories software, hardware, and composition elements. The software elements (*Application Software*) describe the informational structure representation, i.e. the set of elements that provide the application structure allowing for its concurrent running (FEILER; GLUCH, 2012).

The hardware elements (*Execution Platform*) detail the physical components. By the use of these, structural properties related to the real hardware are defined, and based on these characteristics properties can be evaluated and validated such as scheduling, flow latency, memory usage, weight, power consumption, among others. The compositional elements (Composite) are included to support the hierarchical representation. By the use of these components the system

Figure 4 – AADL Components.



Source: Feiler & Gluch (2012)

is described as a set of systems. In this way, the generation of independent systems integrated into the application is allowed.

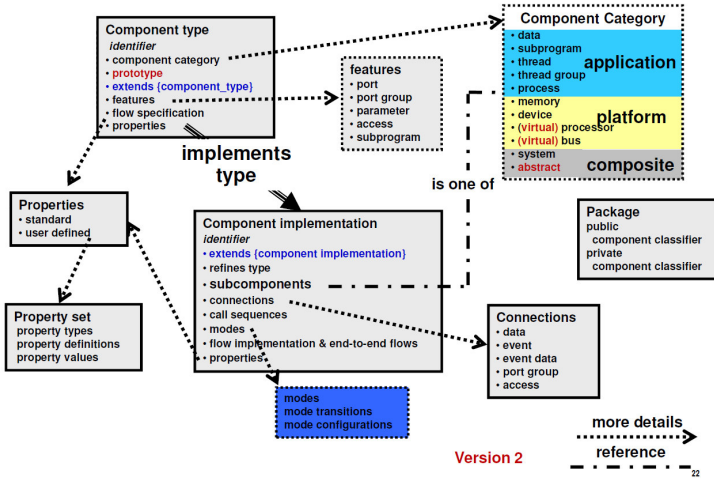
The AADL components are specified based on a set of properties, applied to detail the system characteristics. These properties are summarized in Fig. 5, providing an overview of the relations between components and properties.

As defined in the AADL standard, each component is classified according to category type, represented by component declaration, where the general characteristics are defined. These declarations should be instantiated by the definition of a component implementation. These structures improve the components declaration, providing more information related to component properties.

Regarding component declaration, these structures support the representation of different properties such as input and output ports, subprograms, data flow definition, among others. On the other hand, on the implementation component characteristics such as connections, and operation modes can be described.

The AADL language can be extended by the use of extension paths called annexes. In this way, different properties can be added to architectural representations, in order to detail properties such as behavioral and error characteristics. The extension provided by the behavioral annex enables the representation of the system behavior characteristics, defining properties like its operation modes. On the other hand, by using the error annex, the designers can represent

Figure 5 – Relations between AADL components.



Source: Feiler & Gluch (2012).

the performed behavior in case of failure, aiming to detail the error propagation and cover aspects of architecture reliability.

Aiming to support the system property evaluation, the AADL support the design of system instances, that represent an architectural scenario. By the use of instances the designers can evaluate system properties, ensuring that the designed application meets its restrictions.

Regarding the architectural model design, different tools can be applied to represent its aspects, such as the TOPCASED (FARAIL et al., 2006), and Osate 2 (SEI, 2005). These tools provide a means to represent the architectural characteristics to integrate software and hardware components.

The TOPCASED project is an open-source environment designed to support the modeling process of embedded critical systems. Its structure is based on the eclipse environment, providing a set of tools to support the models construction (FARAIL et al., 2006).

This tool is not only a set of features to provide the integration of different components such as communication, specification, behavior, and architecture in real-time or statical, but is also a set of standards proposed to guide the embedded systems design process. The TOPCASED environment supports the hardware and software specification from design to implementation.

The proposed approach makes use of different design languages, applied according to the performed design activity. The set of tools provided by TOPCASED allows for not only the models design, but helps the designers establish the communication between its representations. This characteristic is possible by the fact that the tool is based on metamodels, that provide the graphical representation and support the transformation processes, generating different representations based on models properties (FARAIL; GAUFILLET, 2005).

Another tool widely used on the architectural modeling is the Osate 2, that provides an environment to design, simulate, and validate architectural aspect of the CPS systems. Considering that this tool was applied to this thesis scope, a section was created to better represent its characteristics.

### **2.2.3 OSATE 2**

The Osate 2 is an open-source tool applied to architectural design, supporting the second version of AADL. By using Osate 2 its possible to represent the architectural aspects in AADL, allowing both the graphical and textual representations (SEI, 2005).

A set of plugins integrate this tool, that is designed in the Eclipse environment (ECLIPSE, 2000). Osate 2 has been under design at Carnegie Mellon university since 2005, the tool structure supports the specification of software and hardware components, providing its integration. Based on model characteristics, application properties can be evaluated, validating the designed system.

Different characteristics can be evaluated by using Osate 2 such as resource allocation, security levels, flow latencies, priority inversion, and scheduling. The resource allocation provides the evaluation of potential processors, memory capability and bandwidth, taking into account characteristics such as task periods, its deadline, processor power, and scheduling police. The security levels analysis enables the evaluation of the predefined security levels applied to each component, evaluating if the defined value is higher between its subcomponents. The system connections are also verified, evaluating whether or not the transmitters have the security level less or equal to its receptors. If any security level is defined, its level is described as zero, the lowest security level.

Regarding the flow latency evaluation, it's possible to analyze

the amount of latencies on each defined flow, analyzing if its value is in accordance with the defined flow. The amount of latency applied to a flow is defined by the sum of its latency connections. On the priority evaluation, priority inversion are analyzed, evaluating whether or not the defined priorities are satisfied during the processor calls on the set of tasks. The scheduling analysis evaluates whether or not the set of defined tasks can be correctly executed, by performing the defined scheduling algorithm (rate monotonic).

Besides functional and architectural modeling processes, the evaluation designed system properties, by the formal methods usage, are an activity applied to the CPS design. By the use of this technique the system characteristics are evaluated and validated.

#### **2.2.4 Formal Verification**

Considering the CPS's inherent complexity, and aiming to ensure that the system requirements are met, additional analysis is required during the design process. In this sense, formal verification becomes a natural candidate to become part of the design process.

Regarding software and hardware design of complex systems, it's observed that more time and effort are spent on verification than on construction. Techniques are sought to reduce and ease the verification efforts, increasing their coverage. In this sense, formal methods offer a large potential to obtain the integration of verification in the design process, in order to provide more effective verification techniques, and to reduce the verification time.

The formal verification is different to the validation process, in the validation process the designers aim to ensure that the system agrees with its propose. In this sense, in order to validate the systems different techniques can be applied such as simulation and testing, providing a means to evaluate whether or not the system can support, the proposed functionality (CLARKE; GRUMBERG; PELED, 1999). However, regarding the design of large and complex systems (with higher amount of elements) this approach becomes unviable, by the fact that applying techniques such as simulation to these representations sometimes we may not be able to path all the existing system states.

In this way, the verification process is defined as a set of techniques that allow confrontation between the system description and its properties. This process its composed by a set of elements



that describe the system behavior, the set of expected properties, and the decision procedure, evaluating if the designed system meets the defined properties.

The verification process aims to determine the system's semantic correctness, i.e., if the system meets the defined properties (ROUSSEL; LESAGE, 1996). Similarly, Bohem (1979) states that the verification is the process that aims to ensure if the system is being correctly built. The performed evaluations using formal verification usually are split in two categories, behavioral and logic.

Regarding the behavioral evaluation two essential elements are identified, the system behavior and the expected system property, these structures are described as graphs to detail the transition system. The decision process aims to establish the equivalent relationship between the designed graphs, when the equivalent relationship is established the analyzed property is considered satisfied (MALCOLM, 1996).

On the other hand, the logic evaluation process states that the expected properties are represented by using logical formulas, defined by the use of temporal logic. Using this approach the operational description is defined as graphic. In this procedure the decisions are made based on the *model checking* (CLARKE; GRUMBERG; PELED, 1999) technique, that performs a search over all execution states of the evaluated model, defining if the property is satisfied to the designed representation. Aiming to apply the formal evaluation different tools are designed to support this process, such as CADP (GARAVEL; MATEESCU, 2001), Uppaal (BEHRMANN; DAVID; LARSEN, 2004b), MRMC (KATOEN et al., 2011) and SELT (BERTHOMIEU; RIBET; VERNADAT, 2004).

In the formal verification world, there are different approaches available such as Model Checking, Theorem Proving (TP), and Runtime Verification. Each method has its pros and cons, namely: Model Checking suffers from the state explosion problem; TP requires highly technical knowledge and despite its latest developments it still faces many automation problems (due to foundational limitations on the supporting logical theories); and RV brings overhead to the CPS since monitors have to be coupled with the components existing in the system. Nevertheless, model checking seems to be the more natural approach for our work since it natively supports the analysis of safety and liveness properties via temporal logics, and it is a fully automated method which is close to the engineering practices and to the abstraction level of models built following the MDE approach.

Aiming to avoid the state space explosion in the Model Checking,

different design techniques can be applied in order to ensure the property evaluation. These techniques include: abstractions and reduction of unnecessary states; the use of symbolic model checking, applying binary decision diagrams and symbolic algorithms; the partial order reduction that concerns the identification of interleaving sequences, eliminating this redundancy reducing the state space.

In order to support the formal verification based on model checking, timed automata should be created to express the system behavior, supporting the evaluation of different properties such as safety, reachability, liveness, and deadlock (BAIER; KATOEN, 2008). However, generating these representations is not a simple task and requires sufficient knowledge of the design team to correctly express the system properties, and therefore there is a clear need for techniques to automate the automata generation process in order to make it less error prone. To perform model generation in an automated way, we believe that model transformation techniques should be applied.

### 2.2.5 Model Checking

In computer science, the Model Checking is defined as an approach that aims to automatically evaluate if a system model meets a specification. Essentially this representation describes hardware, software, and communication protocol components, and its characteristics define requisites such as safety, and deadlock absence (CLARKE; GRUMBERG; PELED, 1999).

Model checking is a verification technique that explores the set of possible system states in a brute-force manner. Similar to a computer chess program that evaluates the possible moves, a model checker, the software tool that performs the model evaluation, examines all possible scenarios. In this way, it's possible to determine if a given system model truly satisfies a certain property (BAIER; KATOEN, 2008).

This technique requires a precise and unambiguous statement of the properties to be examined. To create an accurate system model, the designer needs to lead to discovery of several ambiguities and inconsistencies in the informal documentation.

The system model is usually automatically generated from a model description that is specified in an appropriate programming languages. The property specification prescribes what the system should do, and what it should not do, whereas the model description addresses how the system behaves. The model checker examines the set

of system states to evaluate if the system satisfies the desired property. If a state violates the property under consideration, the model checker provides a counterexample that indicates how the model could reach the undesired state. The counterexample describes an execution path that leads from the initial system state to the state that violates the property being verified (BAIER; KATOEN, 2008).

The designed logics are interpreted over transition systems that describe how a reactive system may evolve from one state to another. Considering synchronous systems, and transition systems, logics such as LTL or CTL can be applied to express timing constraints. Transition system representations of asynchronous systems without additional timing information are indeed too abstract to adequately model timing constraints.

In order to support the formal verification based on model checking, timed automata should be created to express the system behavior, supporting the evaluation of different properties such as safety, reachability, liveness, and deadlock (BAIER; KATOEN, 2008). However, generating these representations is not a simple task and requires sufficient knowledge of the design team to correctly express the system properties, and therefore there is a clear need for techniques to automate the automata generation process in order to make it less error prone.

Aiming to support the systems evaluation by using this technique, different languages and tools should be applied, representing the system behavior, and the set of expected properties such as Lustre (HALBWACHS et al., 1991), SELT/Tina (BERTHOMIEU; RIBET; VERNADAT, 2004) and UPPAAL (BEHRMANN; DAVID; LARSEN, 2004b).

Lustre is defined as a synchronous and declarative language applied to design reactive systems. Its structure supports the design of critical, and real-time systems. This language is defined as declarative, and provides a means to specify a set of equations that are evaluated based on system variables. The synchronous characteristic enables the integration with external events, ensuring their interpretation instantaneously (HALBWACHS; RAYMOND, 2001; HALBWACHS et al., 1991).

Semantic language is defined using a data flow approach, describing the output flow according to received inputs. In this sense, a flow is defined as a sequence of finite or infinite values, associated with a variable, in this way, on each reaction an value is generated (HALBWACHS, 2005).

The State/Event LTL model-checker (SELT) is a tool applied

to formal verification properties, being part of TINA (BERTHOMIEU; RIBET; VERNADAT, 2004) toolbox, and applied to the Topcased project. This toolbox was designed by the French laboratory LAAS, aiming to support design, and analysis of Timed Petri Nets, and Timed Transition Systems (TTS), conducting the model checking evaluations based on logical verification.

The SELT inputs are defined as Kripke structures that describe the system characteristics based on mathematical formalisms, and by using LTL logical formulas, the expected properties to be evaluated are defined. These structures are the result of a system model compilation, represented as a TTS system. The Kripke structures are commonly known as unlabeled transition systems, being represented as directed graphs that specify the set of reachable model states.

The SELT provided verification results may be stored on a text file. In this structure the result of each evaluated property, and if some formula is not verified a counterexample of that violation, is presented. The counterexample can be loaded into the TINA tool to observe the execution path that invalidates the property.

The another tool widely applied to model checking is the UPPAAL, that provide an environment to design, simulate, and validate CPS properties by using formal methods. Considering that this tool was applied to this thesis, a section was created to better represent its characteristics.

### 2.2.6 UPPAAL

The Uppaal tool is an integrated environment that allows users to represent the system behavior in terms of states and transitions, simulating and analyzing the model's results. The tool was designed in collaboration with the Department of Information Technology at Uppsala University in Sweden and the Department of Computer Science at Aalborg University in Denmark. The first UPPAAL version was released in 1995. The tool is available for free for non-commercial applications in academia, and for private persons via <http://www.uppaal.org>. For commercial applications a commercial license is required, see <http://www.uppaal.com> (BEHRMANN; DAVID; LARSEN, 2004a).

Uppaal is a toolkit with a wealth of possibilities to model and analyze systems. The tool was designed to verify systems that can be represented as networks of timed automata extended with integer

variables, data types, user defined functions, and synchronization channels. The main UPPAAL structure is composed of a set timed automata, describing the system behavior, and a query language, applied to express the set of expected system properties to be evaluated (BEHRMANN; DAVID; LARSEN, 2004b).

The tool is composed essentially of three main parts: the system editor, that can be used to built the models; the simulator, in which the system behavior can be simulated and behavior evaluated; and the verifier, in which the system behavior can be analyzed (BEHRMANN; DAVID; LARSEN, 2004a).

Regarding the model's construction, UPPAAL is composed by a set of four different components, describing system variables, system functions, templates, and property queries. The system variables are applied for information exchange, or to support the system design. These variables can have different types, as is commonly on program languages such as C for example. However, The UPPAAL also have some specific variable types such as *chan*, to represent the communication channels between templates; and the *clock*, that is defined to represent time in advance.

To represent the system behavior system functions can be implemented, supporting the system design. These structures may be applied to different components such as transition guards, transitions updates, among others. By the function usage some characteristics can be abstracted from the designed templates, providing a clearer representation.

The system templates support the timed automata representation in order to detail the system components behavior. These structures are composed of a set of states, that represent the execution possibilities, and by a set of transitions that define the communication paths between the system states. To each designed template an initial state is required, and a set of properties can be associated with these structures such as, urgent and committed properties, and associated invariant. The defined transitions should also contain a set of information associated such as: a restriction guard; an update expression, to be executed on the transition occurrence; and an synchronization, where based and defined channel the transition can be activated.

In order to provide the systems evaluation, queries need to be created to represent the expected system properties. To support this specification the UPPAAL uses a simplified version of Timed Computation Tree Logic (TCTL)(ALUR; COURCOUBETIS; DILL, 1990).

In this way, the designer translates the defined restrictions and properties from natural language to formal representation, in order to perform the system property evaluation.

The UPPAAL tool supports the probabilities definition associated with it's transitions. Based on these specifications, and supported by the SMC (DAVID et al., 2015) tool, evaluations can be performed seeing these probabilities. In this sense, characteristics such as the probability of a state being reached or activation of a system behavior can be evaluated with different levels of confidence.

The property evaluation describes the execution of MC itself, i.e., the designed models are submitted to the evaluation of the defined queries, analyzing the produced results. If a defined property is not satisfied a counterexample is produced, allowing that the designers evaluate the situation, and if possible they adjust the system to satisfy the property. Based on the defined queries different characteristics can be evaluated such as reachability, liveness, safety, deadlock freeness among others.

## 2.3 SUMMARY

In this chapter different technologies and techniques applied to CPS applications design are presented. These components include the MDD, concepts of models, metamodels, and model transformations, and a set of tools and languages applied to this propose. Based on these technologies different projects, and complementary representations are generated in order to ensure the applications characteristics specification.

Regarding CPS complexity, especially when considering the UAV applications, its design process is not a simple task. In this way, different techniques, languages, and tools may be applied aiming to support the systems characteristics representation. In this sense, some of the presented components in this chapter are applied to this thesis scope including MDD, Matlab/Simulink, AADL, Osate 2, Model Checking, and UPPAAL. These items are used as a basis to propose a method applied to CPS design and support this thesis construction.

Considering the CPS design process, despite the existing languages and techniques some additional components are required in order to support the application design. In this sense, the next chapter presents some existing approaches, that based on presented technologies provide a means to represent the applications characteristics.

### 3 STATE OF THE ART

Design CPS applications is a complex activity that requires experienced designers to represent the system's characteristics correctly. Regarding the UAV applications, its design process needs to cover several characteristics aiming to engineer aircrafts that fly safe and fulfill the planned missions.

To deal with the complexity design methods are constituted, aiming to guide the teams during the application development. These methods are usually supported by the MDE and suggest using complementary models and tools that are applied to support these representational designs. These tools usually automate design process steps making its activities less prone to errors.

Regarding the design activities, although the authors propose different set steps, some activities are present in most of the approaches, such as functional modeling, design of system architecture, integration with system devices, property verification, project electrics among others. However, despite being present some of these activities do not present sufficient characteristics to ensure its application by the design teams.

Aiming to provide an overview related to the CPS design, in this chapter a set of related works is presented. This set of works was defined aiming to highlight the scientific contributions related to CPS design methods that were proposed over the last years, evaluating and discussing its characteristics and properties. These works are organized into three categories that describe CPS design methods, the integration of sensors and actuators and the formal verification process.

The three presented categories are defined based on contributions to this thesis. In doing so, we aim to provide an overview related to these topics, describing some essential characteristics on each topic to correctly provide the CPS projects design, mainly applied to the UAV design. The works are selected after an exhaustive search on the main scientific databases including [www.scielo.org](http://www.scielo.org), [www.springer.com](http://www.springer.com), [www.elsevier.com](http://www.elsevier.com), and [www.ieeexplore.ieee.org](http://www.ieeexplore.ieee.org).

#### 3.1 RELATED WORKS EVALUATION CRITERIA

Aiming to provide a better work overview, a set of evaluation criteria was defined for each work category. The next subsections detail

the set of criteria specifying the objectives with its definition. The obtained results are showed in the Table 1, Table 2, and Table 3.

For each evaluation criteria, two different symbols are defined, describing respectively the full compliance of the evaluated criteria ( $\checkmark$ ), and the partial compliance of the criteria ( $\diamond$ ). The absence of a symbol defines that the work does not satisfy the proposed criteria.

### 3.1.1 Criteria for CPS Design Methods

In recent years different approaches were proposed to guide the design teams to better lead with the CPS design challenges. This was done with the objective of providing a better evaluation and discussion related works presented, a set of criteria was subsequently defined.

The set of defined criteria includes a *requirements definition*, *functional modeling*, *architectural modeling*, *physical modeling*, *integration of sensors and actuators*, *formal verification*, *system failures definition*, and *hardware and software integration*.

The *requirements definition* evaluates the representation of CPS functional and non-functional requirements on the proposed approach. The *functional modeling* observes the representation of the system functionalities, such as control systems for example. The representation of the embedded architecture to support the functional model is observed on the *architectural modeling*.

The *physical modeling* evaluates whether or not the authors propose activities to represent the system dynamics during the design process. The *integration of sensors and actuators* are with regard to the specification and integration of a set of devices on the application structure. The evaluation of CPS properties using formal methods is observed on the *formal verification*. The integration of hardware and software elements is verified in the *hardware and software integration*.

### 3.1.2 Criteria for Integration of Sensors and Actuators

To better evaluate and discuss the integration of sensors and actuators on the embedded applications the following evaluation criteria are defined including: *hardware abstraction*, *software and hardware mapping*, *detailed devices specification*, *validation and tests*, *software interface*, and *requirement of a Real-Time Operating System (RTOS)*.



*Hardware abstraction* evaluates the use of an abstraction layer to provide isolation between the embedded application and the hardware components. This layer establishes the required information to provide the device's interface. This topic also evaluates how the devices can be integrated into the application architecture, i.e., analyzes if the authors provide information related to the integration of the components on the embedded application. The integration between hardware and software elements is observed in *software and hardware mapping*, which analyzes how the software structures (functions, threads, device drivers) are organized in order to be executed on the embedded platform. The organization related to the thread's execution and the functions to access the system devices on each thread is also observed.

The *detailed devices specification* covers the addition of detailed information related to the specification of these components, evaluating whether or not the described characteristics are enough to provide its interface. *Validation and tests* relate to the analysis of the mechanisms or methods to evaluate the behavior of the integrated devices, both physically and on the device driver level. The *Support execution on embedded platform* requires an embedded OS or RTOS to support the proper application execution, as well as providing the device's interface.

### 3.1.3 Criteria for Formal Verification

To evaluate the presented formal verification approaches the following criteria are defined including: *behavioral characteristics coverage*, *error properties coverage*, *reachability*, *safety*, *liveness*, *deadlock*, *timing properties*, and *statistical analysis*.

The *behavioral characteristics coverage* (*behavioral coverage*) evaluates the CPS behavior representation on the proposed approach, and the translation of these characteristics to the target model. The *error properties coverage* (*error coverage*) observes the error property representation, and its influence on the CPS behavior. The others evaluation criteria analyze the coverage of the proposed methods to ensure properties such as *reachability*, *safety*, *liveness*, *deadlock freeness*, *timing properties*, and *statistical analysis*.

## 3.2 CYBER-PHYSICAL SYSTEMS DESIGN METHODS

Regarding the CPS design, different methods have been proposed over the last years. These approaches are usually composed of a set of activities or steps aiming to guide the application construction. Different coverage levels in the works can be observed i.e., some works are proposed covering the whole design process, while others are more focused on detail specific phases of the CPS design process. The presented works are based on MDE to represent the characteristics of the applications.

### 3.2.1 Overview

Different authors propose the definition of a set of activities supported by the construction of complementary models Jensen, Chang & Lee (2011a), Correa et al. (2010). These approaches propose a set of activities to guide the complete design process covering the main aspects of CPS design that include at least the physical and architectural representation, and functional models to describe the application characteristics. Coupled with the model's constructions, system simulations, tests and verification processes are proposed to validate the systems properties.

Some of these works have focused on detailing specific phases of the CPS design, proposing activities to represent CPS subsystems characteristics such as design of the system architecture, programming the embedded system, among others. In this sense, authors such as Chandhoke et al. (2011), Harrison et al. (2009) present works related to the design of CPS architectural characteristics, focusing on programing these systems. In these works, the architectural representation is translated to the embedded platform by the automated code generation.

Challenges related to analysis and representation of CPS characteristics are discussed in Derler, Lee & Vincentelli (2012), where the authors address the difficulties related to the system characteristics definition, and its dynamic representation to generate the embedded application. The use of complementary models is proposed to deal with the CPS challenges and better represent the systems properties.

Other approaches include the definition of requirements and the capture of project requisites (DOERING, 2014; JENSEN; CHANG; LEE, 2011a; CORREA et al., 2010; MASIN et al., 2017). In Doering (2014)

besides the requirements definition, the design of a PIM and a Platform Model (PM) are also required. Performing a transformation process, these models are merged generating the target application (PSM).

The Cross-layer model-based framework for multi-objective design of Reconfigurable systems in uncertain hybrid environments (CERBERO) project is proposed in (MASIN et al., 2017). This approach aims to develop a design environment for CPS to provide a cross-layer model based approach to describe, optimize, and analyze the system and all its different views concurrently. CERBERO provides: libraries of generic Key Performance Indicators for reconfigurable CPSs in hybrid/uncertain environments; formal and simulation-based methods; a continuous design environment guaranteeing early-stage analysis and optimization of functional and non-functional requirements, including energy, reliability and security.

Works based on the functional model represent the system dynamics on the project design. Based on this representation different artifacts can be generated by performing transformation processes, increasing the project information providing more detailed applications (BRADE et al., 2010; DELANGE et al., 2010; MORELLI; DI NATALE, 2014; DI NATALE et al., 2014). In Brade et al. (2010) the functional model is used as the base for code generation of the target platform AVR. Delange et al. (2010) propose the integration of this model with the architectural model, translating the validated architectural and functional models to code and provide the embedded application. Regarding Delange et al. (2010), Morelli & DI NATALE (2014) these authors propose the generation of the system architecture to support the functional model execution. These representations are based on SysML/Marte to specify the system architectures, as well as a code generation process that is performed to provide the FPGA hardware integration and perform the system execution.

Specifying and integrating hardware components during the CPS process is another design challenge, where the designers need to define a set of devices coupled with the embedded platform to support the application execution. In this sense, a method to specify and integrate hardware components for small-scale helicopters is presented in (CAI et al., 2008b). The authors propose a set of activities to guide the teams to find a better components arrangement considering structural and electrical characteristics such as vibration, and electromagnetic interference, avoiding these effects on the embedded system.

Approaches focused on the design of specific applications are also proposed in the scope of CPS Design. In particular, regarding the UAVs

design in (WANG et al., 2011), the authors propose a method for avionic systems based on ARINC653. This approach describes the design of the architectural model to meet the AIRNC 653 requirements.

### 3.2.2 Evaluation and Discussion

The evaluation of the proposed criteria in respect to the CPS design methods are analyzed in Table 1.

Table 1 – Evaluation of CPS design works.

Evaluation Criteria  Related Work	Requirements Definition	Functional Modeling	Architectural Modeling	Physical Modeling	Integration of Sensors and Actuators	Formal Verification	Hardware and Software Integration
Cai et al., 2008b				✓	✓		✓
Harrison et al., 2009			✓				◇
Brade et al., 2010		✓			✓		✓
Delange et al., 2010		✓	✓	✓	◇		◇
Correa et al., 2010	✓	✓	✓	✓	✓	✓	✓
Chandhoke et al. 2011			✓				◇
Jensen; Chang; Lee, 2011	✓	✓	◇	✓	◇	◇	◇
Wang et al., 2011			✓				◇
Derler; Lee; Vincentelli, 2012		✓	✓	✓	◇		◇
Doering, 2014	✓	✓	✓				✓
Morelli; Di Natale, 2014		✓	✓	✓			◇
Natale et al., 2014		✓	✓	✓			◇
Massin et al., 2017	✓	◇	◇	◇	◇	◇	◇
Design Method for UAVs (Chpt. 4)	✓	✓	✓	✓	✓	✓	✓

A methodology related to the design of small-scale helicopters is presented in (CAI et al., 2008b). This approach is focused on the representation of the embedded hardware, detailing the component

selection, virtual design, definition of embedded hardware, component integration, as well as the ground and flight tests. The authors propose a set of activities related to the hardware project. However, the software characteristics required to perform flights are not detailed, and neither are aspects such as how to define the required devices, and how to better select the components.

An MDE approach based on modular semantics and concurrent languages is proposed in (HARRISON et al., 2009). This method is based on cheap threads<sup>1</sup> that define the task properties. These structures are translated to VHDL language and executed on FPGA hardware. An intermediary language (FSMLang) is used in this process to generate code for the Xilinx Microblaze FPGA. Besides the design of threads and the code generation, other CPS design characteristics are not covered by the authors such as the functional modeling, and the architectural model design. The integration of the system devices is not described, and this process is only based on the FPGA support. The use of formal verification methods is also not defined in the proposed approach.

Brade et al. (2010) describe the design of embedded systems by the use of Model-Driven Design (MDD). This process is supported by a framework named FAMUSO that supports the systems design based on the construction of XML specifications for the embedded platform, and for the set of sensors and actuators. These models are submitted to a transformation process generating a functional model in Simulink. The code for the AVR target platform is generated by using the proposed framework.

Analyzing (BRADE et al., 2010) one can observe that, despite the functional modeling and the sensors and actuators integration, these representations are restricted to the XML constructions. In this sense, the integration of the devices on the embedded platform is not detailed. The definition of an architectural model is not discussed in this approach, not allowing for the evaluation of temporal requirements and others characteristics. Another aspect not presented is the definition of the application system requirements.

The design of architectural and application models integrated with the implementation process is presented in (DELANGE et al.,

---

<sup>1</sup>Cheap Threads is a collection of routines for implementing synchronous threads. It includes a scheduler and a facility for passing messages among threads. Threads may run in a round robin or according to a priority scheme. Each thread must voluntarily relinquish control from time, via an ordinary function return, so that other threads can run. Since the threads are synchronous, i.e. they don't interrupt each other, they don't need to use semaphores, mutexes, critical sections, or other facilities to keep from interfering with each other.

2010). The authors propose the model's construction and validation separately, and these representations are translated into code to be integrated, generating the embedded application. Analyzing the proposed approach it is observed that besides the architectural model representing the set of devices and the application model, the system behavior, the authors do not detail how the sensors and actuators are integrated into this structure. It is also verified that the integration of generated code on the target platform is not clearly defined, not mentioning the requirements of the embedded software to correctly run the designed code.

Aiming to guide the design process, in Correa et al. (2010) the authors propose a methodology that provides a set of steps to guide the teams in the application construction. This approach describes the requirements definition, the functional modeling and simulation, the environmental description, the design of software architecture, the architectural hardware description, the software and hardware mapping, the architectural simulation, the refinement of real-time properties, and the timing verification.

Based on Correa et al. (2010) it is possible to observe that the functional modeling does not detail the integration of sensors and actuators, i.e., the characteristics of the devices need to be manually extracted by the designers based on the physical model, and manually expressed on the architectural model. Besides the proposed hardware and software mapping, the device's integration with the software components is not detailed.

In Chandhoke et al. (2011) the authors have focused on programming the CPS, substantially covering the software architecture design that can be centralized or distributed. The software and hardware integration is partially covered by the generation code process, as well as the clock synchronization properties aiming to manage and support the application execution.

The weakness related to (CHANDHOKE et al., 2011) is the only focus on programming CPS, not detailing the others required CPS characteristics to its design process. Considering the programming methodology it is observed that details related to the sensors' and actuators' integration on the system architecture are not presented, and other CPS subsystems are detailed such as control systems, for example.

Jensen, Chang & Lee (2011a) define a CPS design method devoted to guiding the CPS applications design. The method is composed of ten complementary steps that describe different project

phases. These phases start with the state of the problem, modeling of the physical process, and characterization of the problem. The control algorithm is then derived, the model of computation is selected, and the hardware specified. The system's evaluation is based on simulations. The application is built, the software synthesized, verified, validated and tested.

Besides the approach proposed in (JENSEN; CHANG; LEE, 2011a) covering almost the whole design process, the integration of the system's devices is not detailed, the authors only remark that these components need to be specified. The integration of formal verification methods is also cited. However, the proper integration of these methods on the approach is not described. The authors do not cite the architectural model construction, only citing the use of the MoC, but not describing the integration of these models on the architectural representation.

Wang et al. (2011) propose a methodology to design avionics systems based on ARINC653<sup>2</sup>. This approach is based on the AADL behavioral annex that details the task's behavior. An AADL model named AADL653 was created as an abstract representation of these applications, aiming to support the systems design. The application model details the software structure, and the execution model specifying the hardware components to support the system execution. The avionics application's design is based on the model's integration.

Analyzing the approach proposed in (WANG et al., 2011) it is observed that the method has focused on the representation of the architectural characteristics supported by AADL. In this way, the integration of the software and hardware components, the design of control system, as well as the representation of the system dynamics are not detailed by the authors.

An approach that uses multiple models to represent the CPS properties is presented in Derler, Lee & Vincentelli (2012). The authors propose the use of different techniques to represent the CPS' properties including the representation of its MoC, the use of abstract semantics, actor-oriented models, the hybrid systems, and the system heterogeneity. Both the the design of functional models using Ptolemy, and the system architecture based on MoCs are proposed. The devices

---

<sup>2</sup>ARINC 653 (Avionics Application Standard Software Interface) is a software specification for space and time partitioning in safety-critical avionics real-time operating systems (RTOS). It allows the hosting of multiple applications of different software levels on the same hardware in the context of an Integrated Modular Avionics architecture.

integration is verified only on the design of the physical model that is integrated with the control system. However, this integration is not detailed by the authors.

In Derler, Lee & Vincentelli (2012) it is noticed that the sensors' and actuators' integration is only based on the physical and architectural models, detailing neither the characteristics related to the devices' properties nor the method of how to integrate these structures to the selected MoC. Another weakness is related to the integration of a formal verification process that is not predicted by the authors.

An MDE approach to design embedded systems named HIPAO2 is presented in (DOERING, 2014). This method includes the definition of functional (FR) and Non-Functional (NFR) requirements. The design of a PIM and a PM model represents the application characteristics, and these models are merged during the design process to generate the PSM. The PSM is submitted to a transformation process to provide the code generation for the target platform.

Analyzing Doering (2014), some weaknesses may be observed, like the fact that the functional modeling is based only on PIM design, not detailing how the model properties can be validated. The PM details the target platform's characteristics, however it does not describe how to manage the set of tasks or how to include the scheduling policies on the application. The integration between hardware and software is restricted to the integration of PIM and PM models not describing how to integrate the sensors and actuators on this approach.

Morelli & DI NATALE (2014) propose a framework that provides an execution platform based on Simulink models. The proposed approach details the representation of the architectural model in SysML/Marte to support the Simulink functional representation execution. In this work, the authors have focused on the generation of the execution system, while approaching neither the hardware components' integration nor the use of formal verification methods.

Similarly, in DI NATALE et al. (2014), the authors propose a process focusing on the generation of the system execution platform. This approach also defines the architectural model representation in SysML/Marte based on the Simulink functional model. The mapping between software and hardware is proposed by the use of FPGA components, where the code generating and the usage of abstract interfaces provides the connection between software and hardware structures. However, the component integration is not well detailed by the authors, who provide only a basic process overview, making its reproduction difficult. The use of formal verification methods on this



approach is not cited by the authors.

In (MASIN et al., 2017) the authors propose the CERBERO framework to provide a cross-layer model based approach that supports the CPS design process, some weakness are observed in this approach. The authors describe the support to functional and architectural models, however they do not present details related to how these characteristics are describe in the framework. A MoC is proposed to support the system execution, as well as the integration with the system devices. However characteristics to support this integration between software and hardware elements are not described. Support for formal verification and simulation is also cited by the authors, but the techniques supported by the framework, as well as the type of simulations that can be performed are not described.

Aiming to guide the CPS design process towards facilitating UAV constructions, a design method is proposed on this Thesis and presented in details in Chapter 4. This method depicts a set of phases and activities, covering the design steps and particularities applied to the UAV design.

Based on the characteristics presented in Table 1 and discussed in this section, it is observed that the design process of CPS applications does not have a definitive solution. According to the proposed evaluation criteria, some approaches cover more design steps, providing solutions that can guide almost the complete design process. Other works have focused on detailing a specific project phase and can be used in a complementary way to provide a complete design process. Even though the proposed method targets UAVs design, since such applications is a typical CPS, it can be further generalized to make it suitable to any other similar application design.

The analysis of the mentioned characteristics show that adequately identifying the CPS design phases and proposing activities to guide design teams is an open research area. In this way, works have been proposed to improve the design process and provide more complete solutions. The usage of tools can support the proposed approaches, automating some design activities or steps making the process less prone to errors.

In addition to CPS design methods, evaluation of specific process phases is performed. One of these phases relates to the integration of sensors and actuators on the CPS architectures.

### 3.3 INTEGRATION OF SENSORS AND ACTUATORS

The integration of sensors and actuators on the CPS architecture consists of a set of activities to properly incorporate the system devices on the CPS structure. These activities can include the evaluation of the required devices, the selection of components that fulfill the project requirements on the existing products, and the integration of these different devices into the system architecture.

#### 3.3.1 Overview

It is observed that some approaches are proposed to support designers performing this task, providing solutions with different coverage levels, i.e., some authors propose the integration of these components based only on software aspects while others represent both the software and hardware characteristics, providing the device's interface. In this way, the set of works presented in this section aim to discuss the integration process of these components.

Based on the use of software structures such as shared objects, platform models, Virtual Target Architecture (VTA), and languages such as AADL, SysML, and CPSADL, different authors have proposed approaches to represent the characteristics of the devices, and embedded platform properties (HARTMANN et al., 2011; DOERING, 2014; YU et al., 2013; PAMPAGNIN et al., 2008; SUN; ZHOU, 2013; YU et al., 2011). Hartmann et al. (2011) details the characteristics of the devices by using the SystemC language, while Pampagnin et al. (2008) represent these characteristics using SysML models. Other authors describe the devices' aspects with the design of architectural models (DOERING, 2014; YU et al., 2011; SUN; ZHOU, 2013).

Besides the representation of the system devices, in (YU et al., 2011) the authors provide a MoC to support the system execution. Yu et al. (2011), Sun & Zhou (2013) define the use of an AADL model to represent the device's properties, however, (SUN; ZHOU, 2013) proposing an extension of this language to represent the characteristics of the devices correctly. (DOERING, 2014) proposes the use of UML Marte to describe the device's aspects.

Conquet et al. (2010) propose a tool named *Taste* that provides support to represent the system devices. These components are detailed by using an RTOS, which provides an abstraction layer to support the application design. This layer is responsible for providing the device's

interface. However, the authors do not detail how to integrate these components into the software architecture.

A programming language designed to implement CPS applications named CPAL is presented in (NAVET et al., 2015). The authors propose a C-like programming language coupled with a framework to support the CPS design process. Based on CPAL, devices properties are detailed by specifying characteristics such as the device's interface, the functions for sending and receiving data, among others. An interpretation engine is provided to support the application execution. In this way, the designed application is translated into machine code to run on the embedded platform. Besides the CPAL support of the CPS design and specification of device interfaces, some details are not clearly defined by the authors, such as GPIO mapping for example.

Some authors define a robotic platform as an information processing structure instead of only defining the sensors and actuators in the CPS (CHEN et al., 2015). An architecture named CPSR is proposed, providing a physical layer to integrate the system sensors and actuators properly. Regarding this integration, besides the defined layer, some project's characteristics are not detailed, such as how to select the system devices, and the device's integration with the proper layer. It is also not clear what the designers need to code and what is provided by the CPSR architecture to provide the device's interface.

Ahmed, Kim & Kim (2013) propose an architecture to integrate cyber and physical worlds. This structure is composed of six modules that describe: sensing specification; the application data management; the next generation of the internet; the services aware modules; the application module; and the specification of sensors and actuators. Based on this architecture the authors aim to provide a standard for the CPS design. However, it is not detailed how the set of devices are integrated into this architecture.

### **3.3.2 Evaluation and Discussion**

The evaluation of the set of works that integrate sensors and actuators properties is presented in Table 2.

Regarding the avionic hardware design, in (PAMPAGNIN et al., 2008) the authors proposed an approach devoted to these application's designs. In this way, the construction of SysML representations is proposed to represent the application characteristics. This approach

Table 2 – Evaluation of Integration of Sensors and Actuators

Related Works \ Evaluation Criteria	Hardware Abstraction	Hardware and Software Mapping	Detailed Devices Specification	Validation and tests	Support execution on embedded platform
Pampagnin et al., 2008	◇	◇	◇		◇
Conquet et al., 2010	◇	✓	◇		◇
Hartmann et al., 2011	✓	◇	◇	◇	◇
Yu et al., 2011		✓	✓	✓	
Yu et al., 2013		✓	✓	✓	◇
Sun; Zhou, 2013		✓	✓	✓	
Ahmed; Kim; Kim, 2013		◇	✓		
Doering, 2014	✓	✓	◇		◇
Navet et al., 2015	✓	✓	◇	✓	✓
Chen et al., 2015	✓	◇	◇		
Sensing and Actuation Subsystems Design (Chpt. 5)	✓	✓	✓	✓	◇

defines the generation of multiple models to provide application construction, and some of these models provide an abstraction layer for the application design. Models are also constructed to represent the characteristics of the devices, as well as the integration between software and hardware components being provided by performing a transformation process that generates the application code. However, details related to these transformation processes from model to the application code are not described by the authors.

The ASSERT project (CONQUET et al., 2010) proposes a development process with the focus on the design of complex systems. Based on the MDE principles ASSERT aims to integrate different subsystems as transparently and efficiently as possible. The authors propose the design of multiple models in the engineer’s domain language to represent the application’s properties. These different models are integrated into the TASTE tool to provide the software and hardware mapping. The final application requires an RTOS to support its

execution. To interact with the sensors and actuators, the RTOS must be programmed appropriately. TASTE support the code generation to a set of embedded systems such as Ada runtime, GNAT, and RTEMS, however besides the ASSERT principles its required that the designer addresses some details related to the low-level development, i.e., manually implement architectural aspects such as the device drivers and the mapping pins on the target platform.

The approach presented in (HARTMANN et al., 2011) is based on shared objects to describe the set of system devices. By the design of a VTA, an abstraction layer is proposed to properly interface the hardware components with the shared objects properly. The hardware and software mapping is based on the integration of the shared objects and the VTA. However the authors do not detail what the designers need to perform to integrate these components. Although the devices specification is based on the shared objects described using system C, characteristics like sample time, communication protocol, and ports declaration are not addressed in these structures. This approach supports simulation to evaluate the system characteristics, but the set of properties that can be evaluated is not presented, besides the fact that to run the application n the embedded platform a proper embedded OS is required.

An approach based on AADL is presented in (YU et al., 2011), where the authors propose a CPS design method that includes the AADL and timed automata. On this approach, the characteristics of the devices are based on the AADL components, providing the mapping of the hardware and software components. A verification process is performed based on the generated timed automata, and on the AADL properties to validate the system properties. Despite the proposed transformation process, the authors do not detail its coverage, i.e., what AADL components are used to the timed automata generation. The transformation rules are not detailed, nor are they informed if this process is supported by a tool or manually created based on a designed rules.

An MDE approach named Polychronous is presented in (YU et al., 2013), that provide the functional behavior representation by the Simulink model construction, and the architectural aspects are defined in the AADL model. The architectural model provides the hardware and software mapping detailing the organization of the software structures and its integration with the hardware components. Based on the AADL model the characteristics of the devices are described defining aspects such as sample time, communication protocol, priority,

required access function, input and output ports among others. A model of computation named polychronous multiclock is provided to integrate the designed models and support the system execution, allowing the system simulation and verification. To run the embedded applications, Polychronous provide a proper interpretation engine. However, this engine must be supported by an embedded OS with proper software structures to provide the interface with sensors and actuators and support the application execution.

Sun & Zhou (2013) propose an AADL extension to provide the CPS architecture representation. The CPSADL adds physical and interaction entities to the AADL to better represent the architectural aspects. Discrete computing and continuous physical processes are modeled by the unified CPSADL. The physical process details the sensors and actuators property representation. The software and hardware mapping are provided based on the AADL properties, as well as that the language provides a means to evaluate the system characteristics. Although AADL supports the representation of the architectural characteristics, the authors do not detail the set of properties that can be specified by using this extension. It furthermore details how the language extension details the application property specification.

An architecture to support the CPS design is detailed in (AHMED; KIM; KIM, 2013), that is composed of six modules to represent the CPS characteristics. The sensors and actuators module is integrated into this structure defining the characteristics of the devices as well as integrating of the proposed modules and describing the hardware and software mapping. With the proposed architecture the authors aim to provide a standard for the CPS systems. However, it is not detailed how the set of devices are integrated into this architecture.

Doering (2014) proposes the use of PIM, PM, and PSM models to represent the CPS characteristics. By the use of the PIM, an abstraction layer is provided, allowing for application design without concerns related to the embedded platform. The PM provides a means for representation of the hardware characteristics. However, the authors do not detail what kind of device properties can be defined. The integration of the PIM and PM models is provided generating the PSM model. However, these processes are not well detailed. Characteristics related to the evaluation of the proposed architecture are not presented, and details concerning the execution of the PSM model on the embedded platform are not discuss.

The CPAL is devoted to modeling, simulation, and verification

of CPS (NAVET et al., 2015). The proposed language provides an abstraction layer for the applications construction, i.e., it describes both the functional behavior and the system architecture (i.e., how the functions are arranged, activated, and the data flow among them). Regarding the software and hardware mapping its properties are supported by the provided language, as well as, the characteristics of the devices also being represented using the CPAL. The provided tool supports simulations to evaluate the applications, and a real-time execution engine supports the application running on an embedded platform. To properly interact with sensors and actuators, such an execution engine must be appropriately programmed. So far, CPAL only provides a complete engine for the Freescale FRDM-K64F board. Besides its coverage of limited spectrum of boards, the CPAL design principles allow the designer address the intricate details regarding embedded programming (e.g., pin 0 of GPIO represents the “push button” reading).

Chen et al. (2015) propose the CPSR architecture applied to the robot’s design. This architecture provides a physical layer, that is responsible for the resource management and device interface. An informational layer describes high-level application characteristics. The integration between these layers provides the hardware and software mapping. However, the architectural characteristics definition applied to the system devices as well as these components integration into the system are not detailed.

Regarding the support of CPS sensing and actuation subsystems properties representation, a transformation process is proposed in this thesis (Chapter 5), providing the generation of AADL architectural representation based on functional Simulink models. More details related to this transformation process are presented in Chapter 5.

Based on characteristics presented in Table 2 and discussed in this section, it is observed that to integrate sensors and actuators into the CPS architectures is not an easy task, furthermore according to the design approach different information is required to express these component properties and provide their interface. In this sense, it is observed that this integration process does not have a definitive solution. In this sense, our proposed transformation process aims to improve the representation of the sensing and actuation properties during the CPS design process.

According to the proposed evaluation criteria, it is observed that device characteristics specification and the hardware and software mapping is covered in all of the cited works. However, the presented

information sometimes is not enough to represent these activities. The validation and tests, aims to evaluate the component integration and the application behavior and are not covered in any approaches. Furthermore some authors do not discuss the requirement of the embedded software to support the system execution by sometimes not providing a solution that can be executed on the embedded platforms.

These characteristics demonstrate that adequately identifying the required information to provide the interface to the system sensors and actuators with the embedded architecture is an open research area. In this way, it is essential to correctly identify the required information to provide the device's interface, as well as, allowing the applications to interact with these components receiving measurements and sending references according to the application requirements.

### 3.4 FORMAL VERIFICATION ON CPS DESIGN

Due to the CPS design process complexity, robust system analysis is required to ensure the system requirements. In this way, the MC became a natural candidate to become part of the overall CPS design process, and this process requires adequate tools and methods to support the systems properties evaluation.

In this context, a set of works that provide MC applied to the CPS design are presented in this section, highlighting its characteristics and the set of evaluated properties. The authors provide different approaches to ensure the formal systems evaluation.

#### 3.4.1 Overview

Some approaches are based on architectural an AADL model as input and provide different representations to support the MC evaluation. Yan et al. (2014), Hu et al. (2015) propose the generation of Timed Abstract State Machines (TASM) to support the system evaluation. Correa et al. (2010), Berthomieu et al. (2015) introduce the MC process based on a model transformation chain that uses fiacre as intermediary language and then translate it to Timed Petri Nets that are evaluated by using the TINA tool. Renault, Kordon & Hugues (2009) propose the generation of TPN based on AADL models. However these authors do not use an intermediary language. Furthermore, the selected tool to support the property evaluation is



CPN-AMI instead of TINA. Hamdane, Chaoui & Strecker (2013), Yu et al. (2011) propose the evaluation of the system properties by the use of the UPPAAL tool. In order to perform this evaluation timed automata are generated based on AADL model.

The Networks of Priced Timed Automata (NPTA) generation based on AADL models are proposed in (BAO et al., 2017), aiming to provide the system properties evaluation against various complex performance and safety queries. Other authors also propose the NPTA generation to evaluate the CPS properties (XU et al., 2016). However, its transformation process is not based on AADL, providing its representation based on ThingML language.

A subset of AADL is proposed in (BOZZANO et al., 2009). This language is named SLIM and integrates some behavioral properties (from AADL behavior annex) directly onto the native language. In this way, a transformation process is proposed from SLIM to LTS to support the MC evaluation.

The evaluation of the digital-systems is proposed in (ISMAIL et al., 2015). In this work, the authors formally evaluate the designed control systems. This process is based on the control laws that are integrated into a designed tool to support the properties evaluation.

### 3.4.2 Evaluation and Discussion

The evaluation of the proposed criteria that includes formal verification methods is presented in Table 3.

Renault, Kordon & Hugues (2009) propose the mapping of AADL behavioral semantics into TPN. This process focuses on threads and their interactions with the ports connections. Based on the TPN the designers can evaluate behavioral properties. These properties include deadlock detection, message flow, communication boundness and safety. Time-based properties are also evaluated such as a missed deadline and missed thread activation. The property evaluation is performed by connecting a Petri Net model checker, and defining LTL properties.

Analyzing the proposed work, the main verified weakness is the sole focus on thread behavior, without consideration for any other properties for example reachability or liveness. The authors do not describe the AADL behavior annex usage, characteristics related to the transformation process nor is it made clear whether or not the process is automatically or manually performed. The authors do not cite the

Table 3 – Evaluation of CPS Formal Verification

Evaluation Criteria	Behavioral coverage	Error coverage	Reachability	Safety	Liveness	Deadlock	Timing Properties	Statistical Analysis
	Related Works							
Renault, Kordon & Hugues, 2009	✓	◇		✓		✓	✓	
Bozzano et al., 2009	◇			✓	✓			
Correa et al., 2010	✓		✓	✓	✓	✓	◇	
Yu et al., 2011	✓		◇	◇	◇	◇	◇	
Hamdane, Chaoui & Strecker, 2013	✓		✓		✓	✓	✓	
Yan et al., 2014	✓		◇	◇	◇	◇	◇	
Hu et al., 2015	✓		✓			✓	✓	
Berthomieu et al., 2015	✓		✓		✓	✓		
Ismail et al., 2015	◇			✓			◇	
Xu et al., 2016	✓		◇	◇	◇	◇	◇	✓
Bao et al., 2017	✓	◇	✓	✓	◇	◇	◇	✓
Integrating Formal Verification into the UAV Design (Chpt. 6)	✓	✓	✓	✓	✓	✓	✓	✓

use of AADL EA. However, considering the flow message analysis and mission of thread activation, we can conclude that error characteristics are partially covered in this approach. Also, they do not consider the AADL hardware element properties in this process.

In (BOZZANO et al., 2009) a subset of ADDL language named SLIM is proposed, integrating behavioral properties directly into the native language. To provide the formal verification the SLIM language is translated to LTS, to support the safety and liveness properties evaluation. The transformation process is automated by using a provided tool support. The authors do not define the covering of the timing properties supported in AADL on the SLIM language. In this way, it is not possible to conclude the evaluation of timing properties. The major weakness of this approach comes from the fact that AADL specifications are not fully supported (due to the use of SLIM). Furthermore, error properties are not considered in this approach.

A formal verification approach is proposed integrated into a CPS

design method in (CORREA et al., 2010). In this way, a transformation process from AADL to an intermediary language named Fiacre is performed for further model verification. To support model checking the Fiacre model is compiled into a format for the Tina tool to be verified by using LTL specifications. Tina tool supports the evaluation of properties such as reachability, safety, liveness, and deadlock freeness. Some timing analysis can also be performed by using the AADL model. The transformation omits the hierarchical information of AADL syntax and concentrates on the thread execution and communication. The major weakness comes from not adequately covering the AADL specifications in the transformation process to Fiacre (Not covering the AADL EA).

A transformation process from AADL to timed automata is presented in (YU et al., 2011). An AADL annex was proposed to represent the CPS properties, supporting the transformation process. The timed automata representations are generated based on the AADL structures (native components, and designed annex), and its properties verified by using the UPPAAL tool. However, although the authors propose a new AADL annex, they do not detail the transformation process coverage, i.e., what set of AADL components and characteristics are used as the basis for the timed automata generation.

With this approach, it's not clear how to perform the transformation process, nor is describing the transformation rules or any information related to a tool support clear. Based on AADL and UPPAAL representations characteristics like reachability, safety, liveness, deadlock, timing properties, and statistical analysis can be performed, however, by the fact that the authors do not detail the set of available properties evaluation makes it difficult to describe if all of these properties can be validated by using this approach. The authors define the use of device components. However, they donot indicate the use of AADL EA.

A verification process of AADL architectures by using timed automata formalism is also proposed in (HAMDANE; CHAOU; STRECKER, 2013). In this way, a transformation process is proposed to extract and analyze the temporal behavior of AADL components defined in the behavior annex. The UPPAAL toolbox is used to verify properties such as timing, deadlock absence, reachability, and liveness. However, the authors not present any information related to the integration of hardware components on this structure. Error characteristics, defined by using AADL EA are not considered in this

approach, neither is statistical analysis.

A model transformation approach to provide TASM based on AADL models is presented in (YAN *et al.*, 2014). TASM allows the application of model checking by using the UPPAAL tool. In this sense, properties such as reachability, safety, liveness, deadlock freeness, and timing properties can be evaluated. However, the authors do not detail the verification process in the proposed approach. The major weakness of this approach relates to the fact that hardware elements are not taken into consideration (e.g., Devices) and that there is no support for probabilistic model checking. Regarding the transformation process, it is observed that error properties are also not taken into account.

A methodology applied to translate a subset of AADL language into TASM is proposed in (HU *et al.*, 2015). This AADL subset includes thread components, port communication, behavior annex and mode change. In this approach, the authors have focused on the evaluation of functional properties such as deadlock freeness, state reachability, timing correctness, and resource consumption. A transformation tool AADL2TASM was designed which can be integrated into an OSATE environment to support the transformation process. The weakness related to this approach is the fact that both hardware components as error properties are not taken into account, and probabilistic analysis is not performed. Regarding the system evaluation characteristics such as safety and liveness are not evaluated in this approach.

Berthomieu *et al.* (2015) propose an approach to formal verification systems based on AADL and Tina. In this process, the AADL model is translated to an intermediary language (Fiacre) and then translated to Tina tool to be verified. This approach covers the fundamental AADL properties coupled with the behavioral Annex. Despite the transformation process, the authors do not detail the generated tina models, and they only define the set of transformation rules. Regarding the verification process mainly three kinds of properties are evaluated, the deadlock absence, liveness, and reachability. Evaluating the transformation process, it is observed that the authors do not cover the error annex, as well as not using probabilistic analysis. The authors do not cite the properties evaluation related to the system safety, despite Tina tool supporting its evaluation.

The use of a bounded model checking tool named as Digital System Verifier (DSVerier) applied to verify digital-system implementation issues, aiming to investigate problems that emerge in digital controllers designed for UAV attitude systems is presented in (ISMAIL *et al.*, 2015). This approach provides an SMT-based BMC

approach, for verifying low-level properties of digital controllers. By the fact that authors have focused on the evaluation of a control system, the application's behavior is not covered. By performing these evaluations safety properties of control subsystems as well as checking of timing constraints can be validated. However, by the fact that the authors have focused on the control system, other system properties cannot be validated. Furthermore, the influence caused by an error event is not taken into consideration by the authors. Likewise, probabilistic analysis is not taken into consideration.

Xu et al. (2016) propose a quantitative uncertainty evaluation framework for ThingML-based IoT designs. In this approach, by the use of NPTA (UPPAAL model) and statistical model checking uncertainties caused by external environments can be modeled, as well as, evaluating different performance queries. By using the proposed transformation rules, the extended ThingML designs can be automatically translated into NPTA models, to conduct the quantitative evaluation against specified performance queries.

Regarding (XU et al., 2016), probabilistic analysis can be performed to validate the system characteristics. UPPAAL tool also supports the evaluation of properties such as reachability, safety, liveness, deadlock freeness, and timing properties. However, the authors do not detail the evaluation of this property on this approach. Regarding the error properties, the authors not detail support in ThingML support to the representation of these characteristics, nor do they present its evaluation.

In (BAO et al., 2017) the authors propose a statistical model checking based framework to perform quantitative evaluations of uncertainty-aware Hybrid AADL design against performance queries. This approach is based on AADL, providing a transformation process from AADL to NPTA models. A language extension is introduced in AADL, called Uncertainty annex for the stochastic behavior modeling, enabling quantitative performance evaluation considering uncertain factors caused by physical environments. By the use of statistical model checker (UPPAAL-SMC), its possible to evaluate reachability and safety queries. The UPPAAL tool also supports the evaluation of liveness, deadlock and timing characteristics. However, these properties are not detailed by the authors. Despite the probabilistic analysis, the authors do not present many details related to the error characteristics evaluation, they only describe the evaluation of uncertainties related to the physical environments.

Regarding the support to make formal verification integrated

into the CPS design process, a transformation process is proposed in this thesis (Chapter 6), providing the automatic generation of timed automata based on AADL architectural representation. Based on these representations the systems can be evaluated by using the UPPAAL tool. More details related to this transformation process is presented in Chapter 6.

As can be observed in the presented works, different approaches are proposed to validate the CPS properties by using MC. The primary focus of its approaches relates to the representation and validation of behavioral properties, and only some works cover the evaluation of error characteristics.

Regarding the set of evaluated properties presented in Table 3, we consider that validating these characteristics is essential to a CPS project. In this sense, it is observed that some approaches almost cover all of this characteristics. On the other hand, some approaches just cover part of these properties, providing a partial CPS evaluation and validation. Statistical analysis is another characteristic that is only covered for a subset of these works. In this sense, the proposed transformation process aims to integrate MC into the CPS design process. As result, more reliable (less error-prone) models and systems are created.

In this context, it is possible to conclude that despite formal verification is considered essential to validate the CPS project properties, a definitive solution that can cover all the required evaluation does not exist. In this way, different approaches can be used in a complementary way, enabling the system evaluation to fulfill the application requisites.

### 3.5 SUMMARY AND ADDITIONAL REMARKS

Analyzing the presented topics it is observed that many efforts have been spent over the last year looking for the CPS development process, as well as to the subareas of this development process. This evolution only reinforces the complexity related to the CPS design, and demonstrate that some topics are not widely discussed, nor do they have a definitive solution.

Regarding the CPS design process (Section 3.2), especially considering the UAV applications, it is observed that some works almost cover the whole design process. However, if we consider the UAV particularities, some characteristics are not widely discussed in

existing design processes. In this way, some UAV characteristics are not covered such as the mission definition for example. To correctly map the application's design, the use of multiple design methods is sometimes required, to provide a guideline to the built the applications.

Based on these characteristics the proposal for a new UAV design method proves feasible, providing a solution that can adequately detail the necessary design activities to build the aircrafts to fulfill the planned missions. Also, to adequately cover the complete development process, some specific activities need to be further discussed such as the integration of sensors and actuators, and the formal property evaluation.

Looking at the integration of sensors and actuators (Section 3.3), it is observed that this process usually includes a device description as well as hardware and software mapping. However, the defined information on these activities is not enough to integrate these components. Another observed point is related to the support of embedded platform application execution, which includes the code design to run on the embedded platform, integrating the set of devices. Usually, this process is manually performed by the designers or is partially automated making the software design more prone to errors.

As presented, the integration of sensors and actuators is an open point requiring an increase in discussions related to the integration of the devices on the embedded applications. Besides, evaluating how to test and validate these structures is essential to ensuring the application's integrity.

Formal evaluation of the application characteristics is another point to be discussed (Section 3.4). As can be observed by the authors propose the different tools used to validate the characteristics of the applications. However, with these approaches it has been verified that some authors focus more on a specific property subset, partially covering this process. Besides, the majority of authors mainly evaluate the behavioral characteristics, not considering for example the influence of error events on the embedded applications.

Evaluating the proposed methods we see that further exploration of the influence of error events on the applicant's behavior is necessary. A probabilistic analysis to better evaluate and validate the applicant's is also required. On the other hand, the validation of properties such as reachability, safety, liveness, deadlock freeness and timing requirements are covered in different approaches, and sometimes a complementary approach is required to validate all of these properties in the application.

As presented in this section, despite different authors providing approaches to systematize the CPS design process, either in the general context or in relation to a specific point, this is an area that needs to be more discussed and detailed. In this way, the next three chapters detail these thesis contributions, providing approaches related to the CPS design process, integration of sensors and actuators and the formal properties evaluation.



## 4 DESIGN METHOD FOR UNMANNED AERIAL VEHICLES

Considering the heterogeneity of the design activities performed on the CPS applications design, as well as the process complexity, a structured process is required to design these applications. This process needs to be properly guide by the design teams, addressing the project phases to express the necessary systems properties.

An Unmanned Aerial Vehicle is a typical CPS example, and its design requires collaboration by experts from different fields. Throughout the design, various techniques are employed, such as the mathematical modeling of physical phenomena, formal models of computation, simulation of heterogeneous systems, software synthesis, verification, validation, and testing. Based on the different design techniques previously mentioned it is possible to observe that the design process of these applications is considered complex, requiring the design of different representations to properly describe the application characteristics. In this way, a reliable design method is required to systematize these tasks and to coordinate the project teams.

Contrasting with traditional computer systems design, where the processes usually are deeply rooted in sequential steps, on CPS applications many things happen at once, i.e., physical processes are composed by different things happening at the same time (DERLER; LEE; VINCENTELLI, 2012). Regarding this concurrency and the process heterogeneity, the CPS design process is not an easy task and requires understanding of the joint dynamics of computers, software, networks, and physical processes.

Besides the technical knowledge, the project manager needs to coordinate different teams that are specialized in the specific UAV properties, such as mechanical engineers, software engineers, electronic engineers, aerodynamic engineers, marketing specialists, and designers. A good communication plan is required, as well as a documentation pattern to provide the information dissemination, and avoid that staff replacement does not result in loss of knowledge.

In this context, this chapter presents an MDE method devoted to UAV design. The primary goal of the proposed method is systematizing the design process, covering the planning, development, and implementation phases. These activities are described aiming to guide designers to represent the UAV features, by using complementary models.

The proposed method is highly inspired by integrated product design methods, like for example PRODIP (BACK et al., 2008), which is typically used by mechanical engineers. However, this proposal can be seen as a kind of specialization of this technique for UAV design. This especially comes from the use of MDD techniques, typically adopted in computer science, and also by considering the control and electronic requirements applied to deal with the UAV complexity. The first version of this proposed method was published in (GONÇALVES et al., 2017).

A Bi-rotor UAV design presents the proposed approach in detail, specifying its aeronautical aspects, the related control system, and the required embedded electronics.

## 4.1 PROPOSED APPROACH

This section details the proposed method applied to the UAV design process. Our focus is on representing the required information to develop such kinds of aircraft. In this way, the designers are guided from mission planning to the UAV final construction, covering the main characteristics of these systems.

An illustration of the proposed method is depicted in Fig. 6, describing the method phases, its steps, and design activities. The proposed actions address the particularities of the UAV design process.

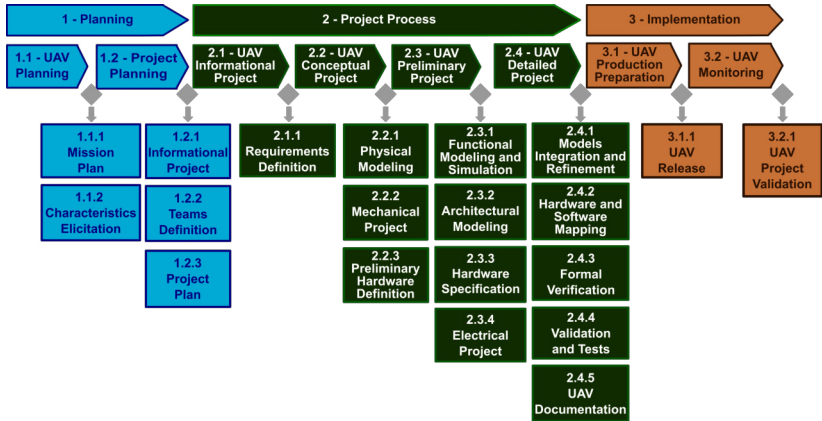
As presented, the method is composed of three phases: *planning*, *project process*, and *implementation*. On each phase, a set of steps are proposed coupled with some activities that aim to provide the project information. The following is the detailed description of the design activities.

### 4.1.1 Design Activities

The proposed design method starts with the planning phase (Phase 1), where the first project features are specified.

**Phase 1 *Planning*:** *In the planning phase the essential UAV characteristics to the started project are specified. The project management information is also performed in this phase, representing features such as communication documents, and the teams management.*

Figure 6 – UAV design method workflow



This phase is composed of two steps, representing the UAV initial definition (*UAV Planning*) and the *Project Planning* (Steps 1.1 and 1.2) respectively.

**Step 1.1 *UAV Planning*:** *This step represents the UAV’s initial characteristics elicitation. In this way, details related to the possible mission(s) that will be performed by the UAV are defined, as well as some aircraft characteristics like weight limit, estimated autonomy, and payload.*

The characteristics of the UAV original definition are normally collected after a set of customer meetings, defining the initial project information. This phase is composed of two activities: *mission plan* and *characteristics elicitation*.

**Activity 1.1.1 *Mission Plan*:** *The mission plan defines the initial UAV description. Simple language is used to represent the mission that resumes the application objectives. This description needs to be clear and direct to guide the project design. As a result of this step, the mission plan that details the application objectives is obtained.*

One can say about this activity that the planning phase can be considered specific to UAV’s design, more specifically in the mission plan is where the designers detail the aspects that the different teams

need to take into account in order to properly design the application. In this sense, the customer provides the mission information in order to details characteristics such as the main UAV objective, the estimated flight autonomy, estimative of size and weight, services that the aircraft need to provide, who will operate the application, environmental characteristics related to the missions, requirement of load transportation, among others. Based on the UAV's mission, the application aspects can be defined. The characteristics elicitation deals with these properties' definition.

**Activity 1.1.2 Characteristics Elicitation:** *The possible UAV configuration analysis is performed in this activity. Moreover, a primary list of devices is defined, coupled with the embedded platforms. Environmental characteristics are also considered in this activity. These properties are defined based on the designer's expertise together with brainstorming sessions.*

Once the fundamental UAV characteristics are defined, some administrative activities need to be performed. These include management decisions to conduct the project's progress, managing the design teams and the product information. These decisions are performed on *project planning* (Step 1.2).

**Step 1.2 Project Planning:** *Management activities are described in the project planning, defining characteristics such as the project's objectives, project time, teams' definition, communication management plan, project constraints, risk evaluation, coding design patterns, and documentation. These characteristics are essential to keep track of the project's progress.*

The project's planning is composed of three activities that define the management actions to start the project. These activities concerns the project's information, teams' definition, and project plan design.

**Activity 1.2.1 Project's information:** *The project's information regards the project's objectives definition, followed by patterns related to the project's documentation and information. The design of document templates, as well as characteristics related to information dissemination are also part of this activity. Regarding the project meetings, characteristics related to its periodicity and involved teams are defined at this time.*

**Activity 1.2.2 Teams' definition:** *According to the project's complexity and characteristics, different teams may be required to detail the application's aspects. In this sense, the project manager evaluates the application aspects and defines the teams according to its expertise.*

**Activity 1.2.3 Project Plan:** *The project plan describes activities to provide the application design plan. To generate this document, estimations are performed, and project characteristics are defined such as project time, project constraints, risk evaluation, resource allocation, and the project definition phase. As output, this activity provides the UAV's project plan.*

With the definition of the UAV's mission, the planning phase is concluded. This information is used to start the *project process* phase (Phase 2), where the UAV characteristics are detailed.

**Phase 2 Project Process:** *This phase details the UAV's characteristics based on the defined mission. According to the MDE principles, different system models are generated in this phase. Moreover, it also addresses model transformation techniques to deal with the project's complexity. As the output of this phase, a detailed project is presented, providing information that allows the UAV prototyping.*

The project process is composed of four steps that go from requirements definition to the complete UAV project. The first step of this phase concerns the *UAV informational project* (Step 2.1).

**Step 2.1 UAV informational project:** *The UAV informational project provides the aircraft's specification. Here the mission characteristics considered represent the system requirements. In this sense, the UAV aspects are detailed, including the system requirements (FR) and the system constraints (NFR).*

This step is composed of one activity that describes the requirements definition (Activity 2.1.1).

**Activity 2.1.1 Requirements Definition:** *Based on the preliminary system description coupled with the mission characteristics, the set of system requirements are defined, to detail the FR and NFR characteristics. This activity generates a requirements document as output.*

Regarding a UAV project, the specification of requirements details system characteristics such as flight autonomy, operating temperature range, power source, operating power source range, actuation limits, collision avoidance features (if required), maximum flight altitude, among others. Based on the UAV requirements, it is required to detail its structural characteristics. This information is expressed in the *Conceptual Project* (Step 2.2).

**Step 2.2 UAV Conceptual Project:** *This step represents the UAV design, i.e., the designers define the aircraft mechanical structure, and its functionalities. These characteristics include the definition of the set of peripheral devices. Different aircraft configurations are evaluated at this time, considering the mission's plan, and the FR and NFR characteristics.*

The provided set of devices at this moment only describes an overview of such components, and this information will be used as the basis for the future detailed specification. To represent the conceptual project characteristics three activities are proposed, defining its physical characteristics (*Physical Modeling*), the detail of the aircraft's mechanical structure (*Mechanical Project*), and providing the preliminary devices definition (*Preliminary Hardware Definition*). The design of the physical model starts in this step (Activity 2.2.1).

**Activity 2.2.1 Physical Modeling:** *The physical modeling defines the system dynamics representation, i.e., a model is created to detail the aircraft dynamics. Mathematical expressions are defined on this model to express the system's behavior in a simulated environment. As output, the physical model is presented, receiving the control references as input and providing the application's behavior as output.*

The physical representation can include environmental characteristics that directly influence the application such as wind, and other disturbances. However, representing the UAV behavior coupled with environmental characteristics is not an easy task, given that the designers usually spend a considerable amount of time to detail these properties using mathematical expressions. After representing the physical characteristics, it is necessary to define the UAV mechanical structures. These characteristics are defined on the *Mechanical Project* (Activity 2.2.2).

**Activity 2.2.2 Mechanical Project:** *The UAV mechanical project regards the application's structural design to properly receive the system*

*devices and the embedded computer. This project aims to integrate the aircraft's mechanical components providing a complete representation. As output, this activity provides the UAV CAD model.*

The aerodynamic characteristics are fundamental to guiding this project, providing an efficient aircraft design during the flights. Additional properties also influence in this project such as shield requirement, weight limit, flight autonomy, and required payload.

Based on the mechanical project a preliminary hardware definition is provided, defining the required device components list. This action is described in the Activity 2.2.3.

**Activity 2.2.3 Preliminary Hardware Definition:** *The preliminary hardware definition provides the set of possible system devices. System characteristics directly influence the components elicitation. If required, some of the selected devices may be replaced on future project phases, as well as adjustments could be required on the mechanical structure to better integrate the system components. These modifications can occur because initially its hard to say that all the selected devices will properly address the UAV objectives.*

The UAV conceptual design defines the aircraft's structure, and the teams can start working on their problems domains. In this way, the UAV subsystems characteristics are provided. This process is started on the *UAV preliminary project* (Step 2.3).

**Step 2.3 UAV Preliminary Project:** *In the preliminary project the set of system modules, responsible for managing the UAV functionalities, are specified. These subsystems perform the interface with the selected devices, providing the UAV architecture. The subsystems are integrated based on their connection ports, providing a high-level system view. As the output of this step provides different representations to represent the complete system architecture, that details the UAV software and hardware structures. This step results into different representations of the system's complete architecture, depicting the UAV's software and hardware structures.*

The *UAV preliminary project* is composed of four activities that describe the functional representation, architectural model, hardware definition, and the electrical project design. These activities are started with the *functional modeling and simulation* (Activity 2.3.1).

**Activity 2.3.1 Functional Modeling and Simulation:** *Design of the UAV behavior representation is performed on the functional modeling, where the control approach is proposed based on the physical model characteristics. The control system is responsible for managing the UAV to meet its objectives. This system design needs to be validated, and its behavior evaluated. The control approach is validated based on simulations, which are performed to measure, evaluate, and adjust the proposed approach. As output, a functional model that integrates the physical model and the control system is provided.*

Analyzing functional design, one can say that proposing a control approach that fulfills the application requirements is a complex task, that involves knowledge of different techniques and promoting an evaluation of which one is recommended to the designed application. Other activities such as system parametrization and simulations also requires high attention by the design team to properly validate the control approach. Sometimes, this parametrization needs to be adjusted when the system is executed on a real environment. This occurs by the fact that the characteristics of the real sensors and actuators are difficult to represent in a simulated environment. A proper software architecture is required to support the control system execution. These structures integrate the hardware and software components. The architecture design is performed on the UAV architectural modeling (Activity 2.3.2).

**Activity 2.3.2 Architectural Modeling:** *The system architecture provides the integration between software and hardware elements. This model provides the mapping processes, and the set of required tasks and functions are defined to support the system execution. The devices interface and the embedded platform characteristics are also described in this representation.*

Representing the physical characteristics such as the embedded platform processing power, available communication buses, system power supply, and devices properties, besides associating these characteristics with the software components is another complex task related to the UAV design. In this sense, a considerable time is spent on this activity in order to provide a proper integration and communication between the application subsystems. The architectural model provides means to evaluate a set of system properties such as system schedulability, flow latencies, and processor usage. Structural characteristics should also be expressed on this representation,



providing means to evaluate physical characteristics like total weight, required power, energy consumption, and the buses utilization.

The set of hardware components need to be selected to integrate with the system architecture. This definition is detailed in the *hardware specification* (Activity 2.3.3).

**Activity 2.3.3 Hardware Specification:** *The hardware specification defines the set of UAV hardware components. Based on the architectural, functional, and mechanical models, the devices and the embedded platform are specified. These components provide the system behavior and perform the UAV actuation according to the designed control strategy. The embedded platform manages the application execution, triggering application tasks according to its characteristics.*

To integrate the system devices into the application structure; its electrical characteristics should be considered. These characteristics are used as the basis to provide the system's power source. These properties are defined on the *electrical project* (Activity 2.3.4)

**Activity 2.3.4 Electrical Project:** *The electrical project concerns the specification of the UAV's electrical properties. This activity is composed of two sub-activities that regard the electrical logic description, and the power source project. As output, this activity provides a complete UAV electrical project.*

The *electrical logical project* defines characteristics related to the system's information exchange, i.e., the logical communication level of each device and the embedded platform are analyzed. Logical level converters should be required to establish the system communication.

The power source project evaluates the input power of the required devices and its power consumption, to scale out the primary power system source. This definition may include the use of renewable energy to provide the primary power source, or to increase the system's autonomy.

This project is directly influenced by the system requirements, defining characteristics such as flight autonomy, and system weight. Providing the proper logical communication between the application components, as well as the required system power supply is a point that requires the designers attention. Characteristics related to the electrical system protections are also considered in this activity.

The definition of the electrical properties ends the *UAV preliminary project*. Based on this information the UAV subsystems are

detailed, evaluated, and validated, providing the sufficient information to the construction phase of the aircraft. These characteristics are defined on the *UAV detailed project* (Step 2.4).

**Step 2.4 UAV Detailed Project:** *The detailed project integrates activities that aims to verify, validate and test the UAV applications. Activities include the subsystems validation and the hardware and software components mapping. As output, this step provides a validated UAV representation.*

The evaluation of the UAV behavior must be performed by the use of simulation techniques (e.g., HIL), and the system requirements must be validated by using formal verification techniques (like MC, or Run-time Verification).

This step provides the required elements to the UAV construction and is composed of five activities. The detailed project starts with the *models integration and refinement* (Activity 2.4.1).

**Activity 2.4.1 Models Integration and Refinement:** *The system components integration provides the UAV with a complete embedded solution. To provide this structure, the UAV subsystems are gradually integrated, evaluating if each new inclusion does not influence meeting the system requirements. If required, adjustments can be performed on the designed subsystems, ensuring the modules integration. As output, this activity provides the integrated software system running on the embedded platform.*

Once the application models are integrated, the designers can evaluate the UAV behavior. However, to provide the complete solution, software and hardware structures need to be mapped. This mapping is performed in the *hardware and software mapping* (Activity 2.4.2).

**Activity 2.4.2 Hardware and Software Mapping:** *This activity describes the organization of functions, tasks, and devices mapping, i.e., the designed functions and tasks are individually validated on the embedded platform, evaluating the system communication and the device's interface. To provide the complete solution adjustments should be required for the functions and tasks. As output, the complete software architecture is provided, integrating hardware and software elements.*

Providing the integration between the different hardware and software components, ensuring their proper interfacing and information exchange is another point that requires especial attention from the

design teams. This comes from the fact that generating the embedded code and validating all the interface with this structure requires different knowledges such as embedded systems design, project of device drives, system scheduling among others. After validating the tasks' behavior, the application needs to be formally validated. This evaluation is performed on the *formal verification* activity (Activity 2.4.3).

**Activity 2.4.3 Formal Verification:** *The formal verification regards the evaluation of the system representation by the use of formal techniques. This evaluation aims to ensure that the system restrictions are satisfied. This process technique can be composed of a set of different techniques, static and interactive, to ensure that all the requirements are fulfilled.*

Once the system properties are verified, the application is evaluated and validated performing the missions. This process is represented in the *validation and tests* (Activity 2.4.4).

**Activity 2.4.4 Validation and tests:** *Validation and tests regard the evaluation of the UAV construction process. The validation activities include stress tests, evaluation of defined system limits, operational tests including long-term tests, and extensive flight tests. At this time the system's properties are validated at runtime, and if required adjustments can be made in the application structure.*

To maintain the project communication, its documentation needs to be updated, providing means for a future project reproduction. These details are presented in the UAV documentation (Activity 2.4.5).

**Activity 2.4.5 UAV Documentation:** *During the project's lifetime different documents are generated representing the aircraft's characteristics, detailing project decisions, and maintaining project information. Compiling this information keeps the project documentation up to date, gathering details of the project cycle, and allowing its reproduction. Operation manuals are also produced in this activity.*

This activity ends the second project phase, and the UAV is ready to be produced and released. These activities are described in the *implementation* phase (Phase 3).

**Phase 3 Implementation:** *The implementation phase represents the final design step, i.e. the UAV is manufactured, validated, and released. Due to the UAV characteristics, usually, these applications are not built in large scale. A single solution is delivered to fulfill the client requirements. At the end of this phase, the UAV is delivered to the customers, and is ready to perform the missions.*

Two steps compose the implementation phase, the *UAV production preparation* (Step 3.1) and the *product monitoring* (Step 3.2), which detail the manufacturing, validation, and monitoring processes. These phases are started by performing the *UAV production preparation* (Step 3.1).

**Step 3.1 UAV Production Preparation:** *Product preparation regards the aircraft construction, and performing experimental tests to validate its structure and adjust the set of subsystems (if required). As output, this activity provides UAV solution.*

Once the aircraft structure is validated the UAV can be released to the customers. In this way, the designed solution can be evaluated and validated by the customers.

**Activity 3.1.1 UAV Release:** *In this activity the released aircraft operates in the real environment. This operation is monitored, and parameters can be adjusted to provide a proper behavior during the execution of the missions. During the customers validation phase, extensive experimental flights are performed, evaluating the complete structure. After the UAV is delivered, the designers just monitor the aircraft behavior during the customers operation.*

To ensure the customers project validation, they evaluate if the aircraft meets the proposed mission. During this step the product is monitored, and technical activities can be performed (Step 3.2).

**Step 3.2 Product Monitoring:** *The product monitoring concludes the UAV design process, describing management activities. This step includes monitoring activities such as customers product validation, product behavior monitoring, and accident evaluation. The monitored characteristics are later analyzed, and corrective actions may be proposed to improve the product quality.*

The product monitoring provides the project validation, where the aircraft characteristics are analyzed evaluating properties like product impact, customers satisfaction among others. Next, details of these activities are presented.

**Activity 3.2.1 Project Validation:** *Project validation is based on the product impact evaluation after delivered. In this way, after the product is in operation characteristics such as customers satisfaction, recall actions requirement, and the preventative maintenance are observed. Once the aircraft is validated its quality is improved, the design teams are dissolved, and the product development cycle is ended.*

The proposed method aims to guide the design teams to plan, sketch, design, construct and commercialize the UAV applications. This method defines a sequence of activities to adequately represent the aircraft properties, providing means for building an UAV that fulfills the proposed missions.

To summarize, the method starts in the planning phase, with the mission definition and some management activities. This initial information is used as the basis to the project process, which details the UAV project and allows its construction. During the design process, several models are created specifying the subsystems characteristics. Also, the project documentation is produced. Finally, the UAV is built, validated, and released in the implementation phase. The aircraft is delivered to the customer, and its operation is validated.

This approach is applied to the Vertical Take-Off and Landing Convertible Plane (VTOL-CP) UAV design, to better illustrate the application of this method. In the next, the performed activities are presented.

## 4.2 DESIGN OF A VTOL-CP UAV

To provide a better perspective in relation to the proposed method, a VTOL-CP UAV was designed using this approach. Details of this method are presented in practice, by performing this aircraft design.

The VTOL-CP UAV design is conducted in the context of the ProVant<sup>1</sup> project created in 2012 at the Federal University of Santa Catarina (UFSC), in partnership with the Federal University of Minas Gerais (UFMG). ProVant's objective is to design a small-scale aircraft to be applied to different contexts.

In the ProVant context, the team starts a new project that aims to design a UAV to be applied in Search and Rescue (SAR) missions. The use of these aircrafts in these kind of missions is important due to

---

<sup>1</sup><http://provant.paginas.ufsc.br>

Figure 7 – Rapid Intervention Vehicle.



Source: Almenara (2017).

the fact that sometimes the rescuers need to access difficult regions in a short period of time, providing proper assistance as quickly as possible.

SAR missions have some characteristics that need to be taken into account on the UAV project. These characteristics include reduced aircraft dimensions, versatility to perform SAR missions, and ease of use.

Reduced aircraft dimension allows its transportation in a rescue vehicle (Fig. 7), providing its integration into coordinated rescue emergency situations. The aircraft's use includes monitoring and surveillance missions, and also transportation of medical loads, with as much autonomy and range as possible.

Based on this information the design process is started, and some initial aircraft characteristics needs to be defined. Next, the description of the project activities are presented.

#### 4.2.1 UAV Method Applied to the project Design

The planning phase (Phase 1) starts with the specification of the required UAV characteristics (Step 1.1), where the UAV mission plan is created (Activity 1.1.1). The mission specified to the UAV on the SAR applications define that the aircraft will provide first support to disasters' victims until the rescue teams arrive. The aircraft needs to be able to perform flights up to 30 minutes. The UAV maximum weight is set at 25 kg and needs to transport 5kg of load. This aircraft needs to be able to maneuver in areas with limited space, allowing access in restricted areas.

These characteristics are used as basis for the definition of the application aspects (Activity 1.1.2). Regarding the required different

UAV configuration maneuverability characteristics are evaluated, and a VTOL-CP UAV was defined as project configuration, allowing the aircraft access restricted areas.

This kind of aircraft has fixed wings, and given this configuration can perform flights both like a helicopter and an airplane. Designing an UAV with a VTOL-CP configuration requires a set of devices including sensors, actuators, and embedded platforms.

A set of required sensors are responsible for estimating the system's behavior: Inertial Measurement Unit (IMU); Global Position System (GPS); Sonar; Pitot Tube; Camera; Laser sensor; and Telemetry radio. Coupled with these components a set of actuators are required to provide the aircraft operation: Servomotor; Electronic Speed Controller (ESC); Motors and propellers.

Defining the embedded platform to support the application's execution is not an easy task. To perform this definition more details related to the UAV structure are required to better choose the VTOL-CP platform. However, at the current project time, a preliminary mainboard can be defined to support the system, as well as a secondary board to perform activities like video processing, base station communication, and others.

Once the initial VTOL-CP characteristics are defined, administrative activities are performed (Step 1.2), including management decisions and the teams assembly. This step is started by performing the project's information (Activity 1.2.1).

This activity defines the VTOL-CP UAV, the primary aircraft objective is to provide support to disaster victims. Document templates are created to record the meeting's information. These documents will be available in a public repository, ensuring the availability of the project information to all members.

Regarding project meetings, based on agile design techniques it was defined that each design team will have small daily meetings (maximum 15 minutes) to the project update. Weekly meetings take place with the project teams to report the project's progress, and to discuss possible problems. Once the informational characteristics are defined, the teams need to be assembled (Activity 1.2.2). Considering the VTOL-CP complexity, and based on the managers previously experience essentially five design teams are required.

The aeronautical team, responsible for the aircraft's design definition considering aerodynamical characteristics. The mechanical team provides the structural project, including the aircraft fuselage and the integration of system devices. The electrical team evaluates

the electrical requirements providing the logical communication, and the power supply. The control team designs the UAV control approach to fulfill the mission. Finally, the software team designs the embedded software structure, and the base station software to support the control system and the mission monitoring.

With the project's information and the design teams defined, the next activity aims the design of the project plan (Activity 1.2.3), ending the first project phase. This activity regards making management estimations related to the project cost, lifetime, required number of designers, among others.

Regarding the project complexity, as well as the aircraft size, evaluations are required to validate the subsystems' properties. These activities analyze project characteristics related to the project risks such as the use of redundant systems, the execution of flight tests in controlled areas, and the exhaustive tests. With the VTOP-CP UAV mission defined and management activities performed, the planning phase is concluded. This information is used as the basis to start the project process (Phase 2). Initially on this phase the UAV requirements are defined (Activity 2.1.1).

To better represent the set of VTOL-CP requirements, tables are used to detail the FR and NFR. In this way, a document is created defining the application's characteristics. To illustrate the requirements definition, two functional requirements and associated non-functional requirements are presented in Tables 4 and 5

Table 4 – UAV control stability requisite

F1 - Stability control during flight.				Hidden ( )	
<b>Description:</b> The VTOL-CP needs to maintain its stability during flights, be these autonomously or remote controlled.					
Non-Functional Requirements					
Name	Restriction	Category	R.T.	Permanent	
NFR1.1 - Maximum lateral inclination	The aircraft cannot tilt more than 20° in the roll axis.	Security	Hard	( X )	
NFR1.2 - Maximum frontal inclination	The aircraft cannot tilt more than 30° in the pitch axis.	Security	Hard	( X )	
NFR1.3 - Audible alert	If the maximum inclination is exceeded, the aircraft must emit a beep until it returns to the inclination limit range.	Interface	Soft	( X )	

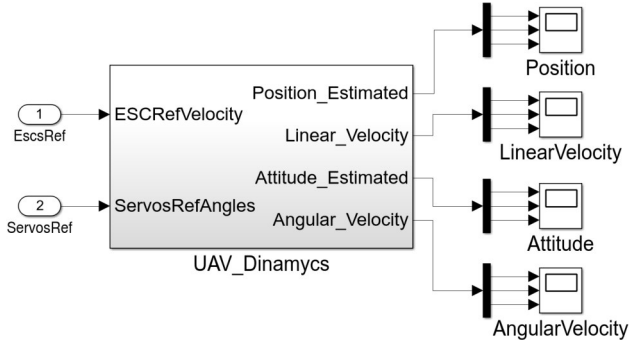


Table 5 – UAV load transportation requisite

F2 - VTOL-CP load transportation.				Hidden ( )
<b>Description:</b> The VTOL-CP needs to transport load during the missions. This load can include medical supplies or a survival kit. The load compartment needs to be easy to access.				
Non-Functional Requirements				
Name	Restriction	Category	R.T.	Permanent
NFR2.1 - Maximum load weight	The aircraft can transport 5kg of maximum load.	Security	Hard	( X )
NFR2.2 - Load access	The load can be accessed only when the aircraft is landed and with the motors off	Security	Hard	( X )

Once the UAV requirements are described, the structural project is started, by performing the conceptual project (Step 2.2). Physical modeling describes the first activity of this step (Activity 2.2.1), representing the system behavior. Based on use of the Simulink tool, the control team design the physical model, that is composed of layers to better represent this structure. The main view of the VTOL-CP physical model is depicted in the Fig. 8, describing the aircraft behavior according to the provided inputs.

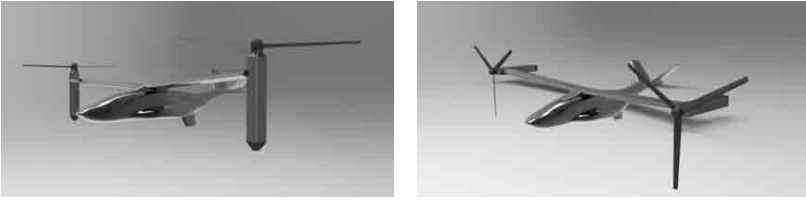
Figure 8 – VTOL-CP physical model.



Based on the behavioral characteristics, detailed on the physical model, the mechanical structure is defined (Activity 2.2.2). In this way, the UAV CAD model is produced, to detail the aircraft structure. Fig. 9 shows the VTOL-CP structure.

A set of devices are required to support the aircraft flights.

Figure 9 – VTOL-CP UAV architecture.

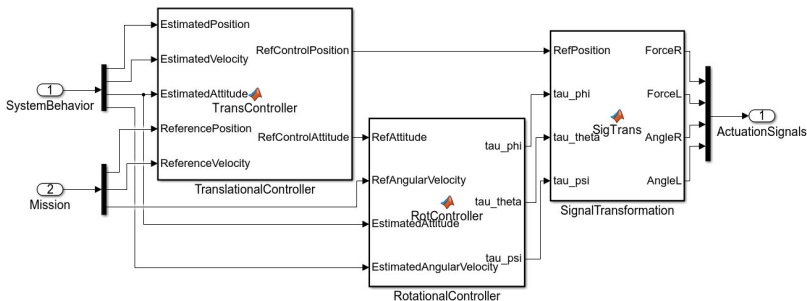


The set of previously defined components (Activity 1.1.2) are revised, evaluating if they can be applied in the designed mechanical structure (Activity 2.2.3). Regarding the VTOL-CP architecture, based on the teams experience at this time the initial set of defined devices are maintained.

With the definition of the aircraft's structure, the teams can start to work on its problems domains, aiming to detail the UAV subsystems. This process is performed in the UAV preliminary project (Step 2.3), and functional modeling starts to be performed (Activity 2.3.1).

The functional model represents the UAV behavior and is designed based on the physical model coupled with the control approach. Fig. 10 depicts the VTOL-CP functional representation, that details the control subsystem, in charge to provide the aircraft stabilization and path tracking.

Figure 10 – VTOL-CP UAV Functional Model.



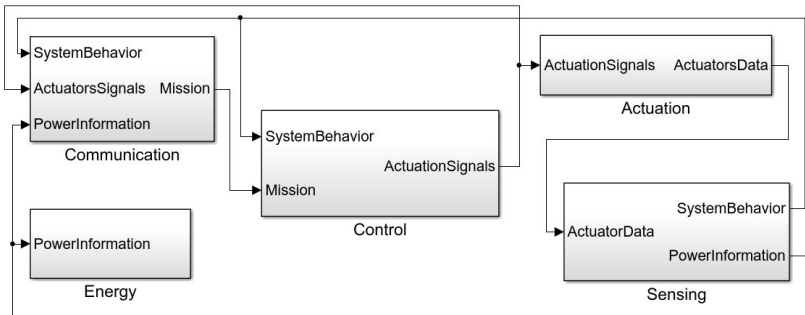
This structure is essentially composed of three blocks: path track control (*Translational controller*), that provide the aircraft tracking ability; aircraft stabilization (*Rotational controller*); and the *Signal*

*Transformation*, that converts control signals into reference signals for each device.

Based on the control approach, a software architecture is designed to support the system execution. This design details the hardware components characteristics such as the embedded platform processing power, the application buses, and devices communication protocols that are integrated with the software structures, being responsible to perform the hardware interfaces and to execute the system functions. The architectural design is performed on the UAV architectural modeling (Activity 2.3.2).

Initially, in this phase, the architecture is represented by using the Simulink tool, providing a representation that integrates the control system to the application subsystems. This representation is depicted in Fig. 11, describing a high-level view of the system's architecture.

Figure 11 – High-level view of the UAV architecture.



The software structure is composed of five subsystems representing: communication, control, energy, actuation, and sensing. The communication subsystem is responsible for establishing a communication channel between the aircraft, base station, and other UAVs. The control subsystem is responsible for managing the system behavior while performing the missions. The energy subsystem provides power control and monitoring of the energy consumption, the use of renewable energy, and the battery recharge. The sensing subsystem is responsible for estimating the aircraft behavior, as well as some additional functionalities, such as image processing and collision avoidance. Finally, the actuation subsystem provides the actuators interface. This structure represents the system modules and their

relations.

The architectural model details the subsystems' properties and represents the integration between software and hardware structures. Based on this initial representation the AADL is used to represent the architectural model. To exemplify, Fig. 12 presents part of the AADL code that specifies the highest abstraction level of the UAV architecture.

Figure 12 – High-view of UAV Architectural model.

```

1  SYSTEM IMPLEMENTATION UAV.impl
2  SUBCOMPONENTS
3  --PROCESS
4  pi_energy: PROCESS p_energy.impl;
5  pi_control: PROCESS p_control.impl;
6  pi_sensing: PROCESS p_sensing.impl;
7  pi_actuation: PROCESS p_actuation.impl;
8  pi_communication: PROCESS p_communication.impl;
9  --DEVICE
10 di_gps: DEVICE d_gps.impl;
11 di_imu: DEVICE d_imu.impl;
12 di_esc_r: DEVICE d_esc.impl;
13 di_esc_l: DEVICE d_esc.impl;
14 di_sonar: DEVICE d_sonar.impl;
15 di_radio: DEVICE d_radio.impl;
16 di_servo_r: DEVICE d_servo.impl;
17 di_servo_l: DEVICE d_servo.impl;
18 di_pitot_tube: DEVICE d_pitot_tube.impl;
19 di_camera: DEVICE d_camera.impl;
20 di_laser: DEVICE d_laser.impl;
21 CONNECTIONS
22 C1: PORT di_gps.position -> pi_sensing.position;
... Here goes all the connections (lines 23 to 48)
49 END UAV.impl;

```

To provide the architectural model generation, a transformation process was designed, supported by the *ECPS Modeling* tool, for the translation of the functional model into the architectural model, i.e., transforming the specification of Fig. 11 to the model detailed in Fig. 12. Please refer to chapter 5 for more information about the mentioned transformation.

The set of hardware components needs to be selected after the system architecture is defined. This definition is detailed in the hardware specification (Activity 2.3.3). The designers perform a market research, evaluating the available solutions, their properties,

and cost. Based on device characteristics coupled with the defined project constraints a set of sensors and actuators is selected to apply to the project structure.

With the system components defined, its electrical characteristics should be considered, taking into account the system's power source. The definition of these features is performed in the electrical project (Activity 2.3.4). This project is split into two sub-activities defining the electrical logic characteristics and the power source project.

The electrical logic activity evaluates the communication levels of each component. Given the fact that these devices usually work with two different logical levels (3.3v and 5v), typical logical level converters are applied to this environment. These components provide support to establish the communication between the system different elements.

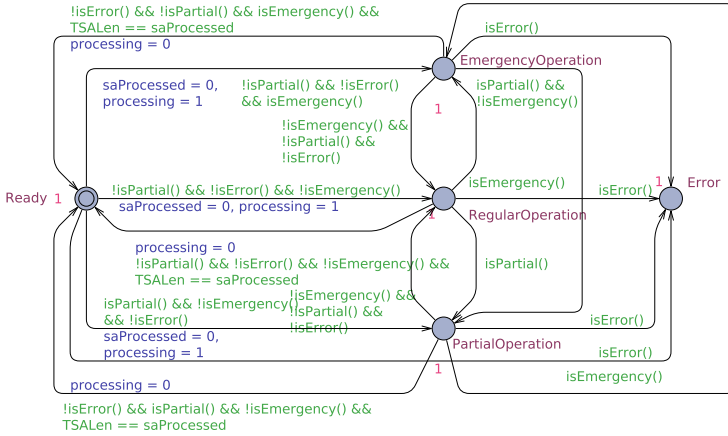
The power source project defines the central system input power. This definition is based on the devices consumption, as well as being influenced by the flight autonomy requisite. The use of renewable energy is also considered in this activity, to provide the primary power source, or to increase the system autonomy. These definitions end the UAV's preliminary project and based on this information the UAV subsystems should be evaluated, and validated. These activities compose the UAV detailed project (Step 2.4). This step starts with the model's integration and refinement (Activity 2.4.1).

To provide the system modules integration in the embedded platform, the designed architectural structure receives each subsystem gradually. In this way, the modules presented in Fig. 11 are sequentially included, i.e. the designed functions and tasks defined to each subsystem are executed on the embedded platform coupled with its required devices. With each new inclusion, tests are performed to evaluate the system behavior. If required, adjustments are performed, ensuring the modules correct integration on the embedded platform.

With the application models integrated, software and hardware structures are mapped aiming to provide the complete solution (Activity 2.4.2). In this sense, the designed functions and tasks are individually validated on the embedded platform, evaluating the communication structures, and the device's interfaces. This activity also evaluates devices characteristics like configuration, response time, and accuracy.

Once the tasks behavior are evaluated, the application needs to be formally validated, where formal methods can be applied to verify the system characteristics (Activity 2.4.3). This process regards the

Figure 13 – Position estimation task behavior.



evaluation of the system properties by the use of formal techniques, ensuring the system constraints are achieved. The assessment can be composed of different methods (static and interactive).

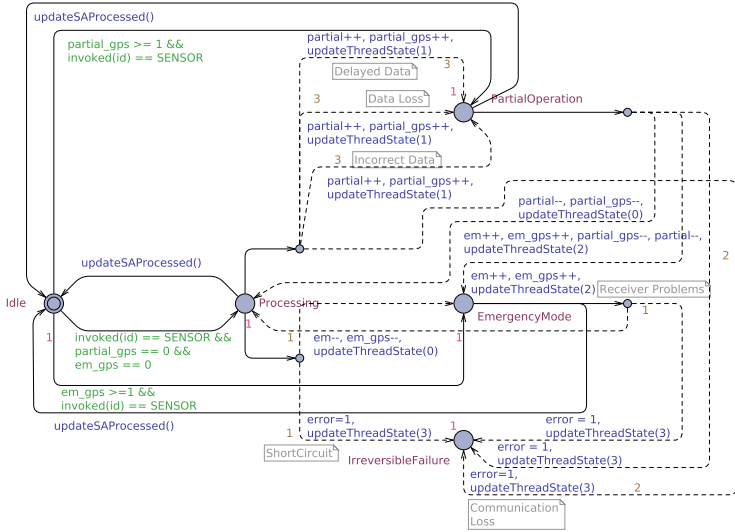
To support the formal verification methods integration on the UAV design process a transformation process was designed, supported by the *ECPS Verifier* tool to extract the timed automata representations based on the architectural model. In this way, the specification presented in Fig. 12 is used as a base to generate the timed automata, evaluated using MC. Please refer to chapter 6 for more information about the mentioned transformations.

By performing the proposed transformation process, the timed automata are extracted based on the architectural model presented in Fig. 12. Fig. 13 details the structure of the position estimation thread.

This thread provides the interface to the GPS device and estimates the UAV position based on this information. In addition to the tasks templates, device's models are generated, aiming to evaluate the influence of these components on the designed task. Fig. 14 depicts a device representation that details the GPS behavioural structure with associated possible failures.

The mission's fulfillment needs to be evaluated after validating the system properties. In this way, tests are performed assessing the hardware and software structures (Activity 2.4.4). In this activity, the VTOL UAV structure is evaluated in order to be released. The validation activities include stress and limit tests, system limits

Figure 14 – GPS behavior representation.



evaluation, long-term tests, and extensive flight tests.

Maintaining the project information providing means to a future project replication is the goal of the project's documentation (Activity 2.4.5). Different documents are generated during the project's lifetime, describing the VTOL-CP UAV characteristics. In this way, this information is compiled, updating the project documentation. At this time operation manuals are also generated.

The documentation process ends the second project phase, and the VTOL-CP UAV is ready to be built and released (Phase 3). This phase represents the final design stage. Due to the UAV project complexity, at this time this application will not be produced in large scale, a single initial solution will be delivered to the clients.

The implementation phase starts with the UAV production preparation (Step 3.1). This step deals with the aircraft construction, where tests are performed on the VTOL-CP UAV to validate its structure, and its composition by the UAV release activity (Activity 3.1.1).

To provide the UAV release the aircraft is operated in the real environment. This operation is monitored to validate its behavior during the execution of the missions. The aircraft is delivered to the designers monitoring the customers VTOL-CP operation, evaluating

the application behavior. Once the aircraft's structure is validated, the UAV is released to the customer's validation. After delivery its operation is monitored (Step 3.2) evaluating of unexpected behavior occurrence during the missions performing.

Product monitoring provides the project validation (Activity 3.2.1), where several project characteristics are analyzed. This validation evaluates the UAV impact after in operation by the customer's. The project is ended after final customer validation and teams dissolution. As described some of these activities are automated by the use of proposed tools that provide not only transformation processes but they also automate some model generation.

### 4.3 SUMMARY

In this chapter a design method devoted to the construction of UAV applications is presented. In this way, a set of phases and activities are presented aiming to cover the particularities of these application constructions. By performing the proposed activities the designers can provide the information required to the UAV's constructions, as well as cover some management activities essential to the projects construction.

Once the propose method was finished, it was possible to observe that the proposed set of steps and activities may be generalized to other CPS application. However, to conduct the method generalization some planning characteristics need to be modified, and a different set of information is required on the mission plan. For such reasons, and also to limit the scope of this thesis, any adaptation or generalization of the current proposal is left for future works.

The proposed approach is applied to the design of a bi-rotor UAV with the tilt-rotor configuration. Based on this case study, it aims to detail the project phases and activities applied to a real application. In this sense, more information related to the performed activities can be presented, as well as its produced artifacts.

Analyzing the proposed method it is observed that the implementation phase was not so widely discussed and that some activities mainly related to the UAV construction could be better detailed. Besides, characteristics related to the costumers product validation are also lacking. Another observed point is related to the application of code generation. Besides this activity is considered implicit in some project phases, generating the application code to



be executed on the embedded platform is not widely discussed on this method.

Whilst aiming to support the proposed design activities some tools are also designed. These applications assist the design teams during the application characteristics representation, and provide the automated generation of a complementary model. In this way, the project can be reduced, by automating some representation constructions, and the project information can be maintained during the whole project time.

In this sense, a tool was proposed aiming to perform a transformation process from functional model to the architectural model. The tool is named *ECPS Modeling*, have focused on the specification of the sensor's and actuator's properties. The details of this transformation process are described in the next chapter.



## 5 SENSING AND ACTUATION SUBSYSTEMS DESIGN

The UAV development process involves complex engineering work. The design of analytical models representing the UAV behavior and its control solution is an essential step within this process (LEE; SESHIA, 2015; ALUR, 2015). This step is typically named as *functional design*, and involves performing simulations to test both the analytical model that represents the UAV behavior and its control solution.

The analytical model representing the CPS behavior is mainly devoted to simulation, so it usually would not be used in the implementation. Designers, however, should not merely discard this model in the final implementation but preserve its structure and adapt it to cope with the sensors and actuators that will be used. While sensors provide information about the application state, the actuators allow executing the control actions to change the device state.

Further, the design team must tackle how to correctly implement the control solution in a typical embedded computing platform. Again, this is not a trivial task, mainly due to the real-time requirements imposed by the control algorithms and the limited computing resources from the embedded platform. This step is typically called *architectural design*, and also involves different types of analysis before what is known as implementation in the target platform.

Regarding the design method presented in Chapter 4, this process is composed of a set of phases devoted to represent UAV characteristics. In this sense, as described in the UAV preliminary project (Step 2.3), the design of architectural modeling (Activity 2.3.2) aims to represent the integration between software and hardware elements. This activity includes the integration of sensing and actuation properties in the architectural representation.

The generation of architectural model that integrates the described characteristics is based on the functional model, where performing a transformation process the designers can extract a set of information from the base model. The transition between these two types of models constitutes the research problem to be addressed in the present chapter.

In this chapter presents some guidelines related to how designers should transform the functional model to remove the analytical representation of the physical device and add the set of sensors and actuators to be used in the implementation, guiding the designers to

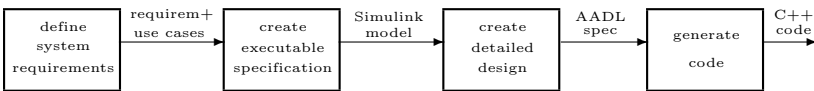
perform such transition. The tool named *ECPSModeling* was developed both to support the mentioned guidelines and also to link the model with specific sensors and actuators (selected devices would come from a pre-defined programming-level repository). The ECPSModeling tool usage is exemplified using a case study devoted to the design of a UAV.

The proposed transformation process detailed in this chapter is defined as a specialization of the UAV design method described in Chapter 4. More specifically the proposed approach describes a set of activities aiming to guide the designers to represent the architectural model (Activity 2.3.2 of the proposed method) and integrate the sensing and actuation characteristics on the designed representation.

## 5.1 RESEARCH CONTEXTUALIZATION

The proposal presented in this chapter is an extension of a previous research work presented in (PASSARINI et al., 2015). This section provides an overview of the model-based development method for CPS design that was presented in (PASSARINI et al., 2015), and that is followed in the present work. This method has four suggested design steps, as follows: (i) system requirements definition; (ii) preliminary design; (iii) detailed design; and (iv) implementation. Fig. 15 depicts the relations between these four steps, including the resulting actions (inside the blocks) and the provided outputs.

Figure 15 – Main activities and artifacts of the method to develop CPS.



From Passarini et al. (2015).

This method suggests adopting different modeling languages to represent systems functionalities and architecture. The reason, therefore, is related to the tool selected to perform simulations of the system functionalities. CPS designers usually prefer using tools that support mathematical modeling and that include simulation capacity, like for example Simulink (the one used in this approach). For representing the system architecture, the adopted development method suggests using a different modeling language, for instance, the AADL.

AADL was selected given that it allows conducting several analysis in the architectural model before its implementation. Therefore, the adopted method requires transforming a functional model described using functional blocks (from Simulink) into an architectural model in AADL.

The core of the work presented in (PASSARINI et al., 2015) concerns the model transformation from functional to architectural specification. More specifically, it presents a transformation engine called AST, which transforms functional (simulation) models designed in the Simulink environment (design step 2) into architectural models represented in AADL (design step 3). While the Simulink model elements represent the target functionalities, the AADL model elements detail a suitable software structure and target platforms for incorporating those functionalities.

An important detail to be highlighted concerning AST is that it does not comprise the generation of the set of sensors and actuators needed by the embedded system. In fact, the specification of such devices should be part of the AADL architectural model. It happens that there is no information about sensors and actuators in the Simulink functional specification, as their designers typically create, for simulation purposes, a mathematical model that represents the system to be controlled. The approach detailed in the next section guides designers to properly bind sensors and actuators into the architectural (AADL) specification.

## 5.2 PROPOSED APPROACH AND RELATED DESIGN ACTIVITIES

This section presents the approach designed to help designers moving from a model that is built for simulation purposes to a model that is devoted to implementation. While the simulation model requires an analytical specification of the physical device to be controlled, the architectural model requires the specification of the sensors and actuators that will be attached to the physical device.

As discussed in the previous section, the present work is conducted as an extension of the method presented in (PASSARINI et al., 2015). Therefore it assumes the use of functional blocks designed in the Simulink tool for representing the functional model and the use of AADL to represent the architectural model. The work described here targets a part of the model transformation

that was not covered in (PASSARINI et al., 2015), i.e., the proposed approach extends the transformation process performed between steps *ii* (preliminary design) and *iii* (detailed design) from (PASSARINI et al., 2015). The proposed approach provides means to represent the sensor and actuator characteristics during the transformation process, generating the architectural (AADL) specification.

An important aspect to be highlighted is that the problem under consideration here is not restricted to merely selecting sensing and actuation devices from a repository and attaching them directly to the control blocks<sup>1</sup>. It happens that the designer must adequately plan how the system should interact with these devices to gather information (in case of sensors), and shape it to provide the required information of the control system. The same applies to sending information from the control to the actuators.

The proposed approach aims to provide means for the designers to specify the application is integrating the software architecture with the system devices. It is important to highlight that this approach is not intended to design the sensors and actuators themselves, i.e., represent their behavior mathematically. The proposed design activities aim to guide the designers on the representation of a proper structure that allows the control system to interface with the system devices. Besides, it also allows representation of some details about how the devices operate.

The proposed approach consists of a set of activities to be conducted by the designer during the transformation process to provide the device's property specifications. The application of the proposed activities is illustrated in a case study devoted to designing the control software of a UAV, as follows.

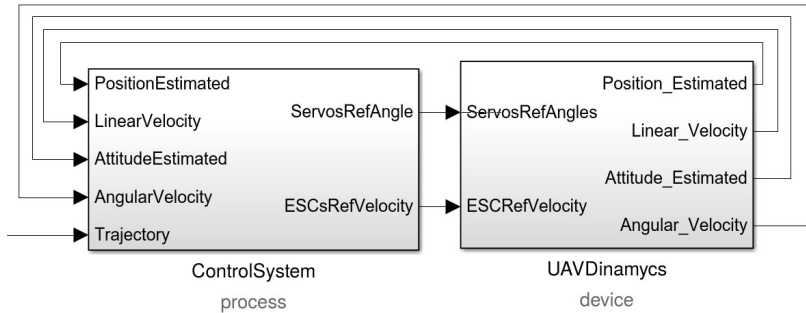
### 5.2.1 Case Study

As presented in 4.2 this project it was conducted within the scope of the ProVant project, whose goal was the construction of an autonomous UAV. It tackled the design of the electro-mechanical components and the embedded control system of the UAV, both hardware, and software. Such a project comprised the design of a Simulink functional model for the control system to be embedded in the UAV and is used as the basis for showing the approach presented

---

<sup>1</sup>*Control blocks* refer to the part of the specification in charge of executing the logic responsible for controlling the physical device.

Figure 16 – UAV Simulink model: first hierarchical level.



in this paper.

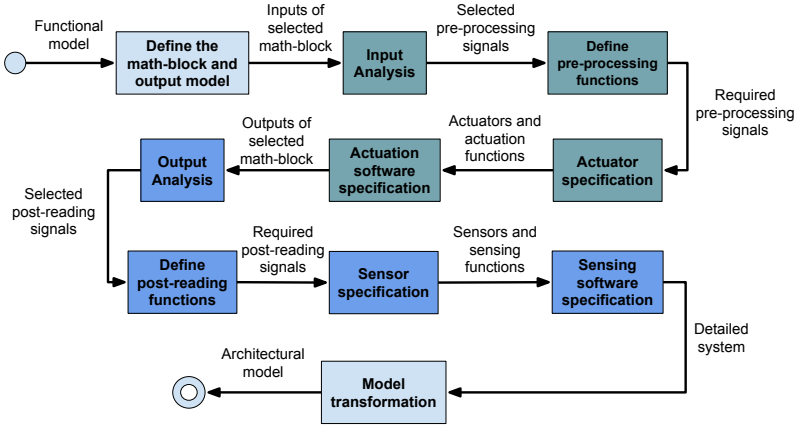
The top-level view of this model is depicted in Fig. 38. The left block represents the control strategy. The right block (*UAV\_Dynamics*) contains a description of the aircraft behavior using a set of mathematical expressions that allows one to extract the aircraft position, attitude, and velocities (angular and linear). These characteristics are estimated according to the received control inputs. The creation of this Simulink functional model allowed the design team to perform simulations aiming to observe the system behavior concerning the control strategy.

Based on this UAV Simulink model, design activities are created to support the specification of the sensing and actuation subsystems. The proposed activities are supported by a tool named *ECPSModeling*. The proposed design activities and the application of these activities on the UAV case study are presented in the next section.

### 5.2.2 Design Activities

The proposed approach consists of a set of activities to be conducted by the designer allowing for provision of the required additional information regarding the sensors and actuators to be adopted so that a proper AADL model can be generated. In summary, the activities allow the designer to select the desired actuators and sensors from a repository, characterize them, and then correctly make their binding into the model. A tool named *ECPSModeling* was developed to help designers performing the proposed activities. It works

Figure 17 – ECPSModeling process workflow.



as a plugin for the *Osate* framework.

The workflow from Fig. 17 depicts the design activities covered by the proposed tool. These design activities are started performing an analysis on the functional model, defining the mathematical block that represents the system behavior. It is then followed by an analysis and further specification of the input and outputs from such model. The input analysis are covered for the second, third and fourth activities (green blocks in the Fig. 17), and the output analysis are performed for the sixth, seventh, eighth, and ninth activities (dark blue blocks in the Fig. 17). It was the authors' decision to start the next step with the actuation subsystem (math-block inputs). However, we do not see any objection to starting with the sensing subsystem (math-block outputs).

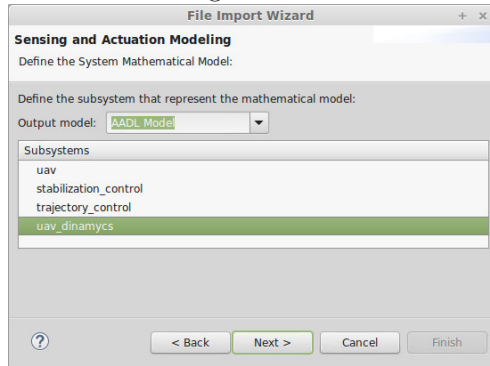
**Activity 1 *Define the mathematical block or device*** - In this activity, the Simulink blocks are listed, and the designer needs to define which component represents the system dynamics (mathematical block). The inputs and outputs of the selected block will be used as the basis for the specification of the sensing and actuation subsystems.

The definition of the mathematical block is depicted in Fig. 18, by using *ECPSModeling* tool. Applying the proposed approach to the UAV design activity 1 is performed selecting the *Uav\_Dynamics* block, which is used on the functional model to represent the system dynamics.

Once the input block is defined, the activity 2 is started. So the



Figure 18 – ECPSModeling definition of mathematical block.



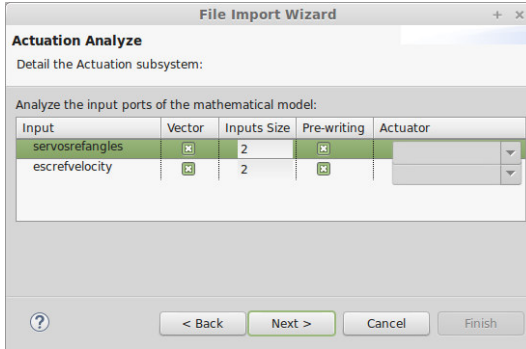
designer needs to analyze the inputs of the selected block, as follows. These inputs regards the control references received from the control subsystem.

**Activity 2 *Input analysis*** - Here the inputs of the selected block are presented to the designer, which needs to characterize the input data with information such as the type and size of data, whether pre-processing is required or not, and the actuator that will consume the data. But the actuator can only be selected if no pre-processing is required, because if the selection occurs it means that the selected input data cannot be directly sent to an actuator, thus requiring some kind of pre-processing to provide the correct information to the system actuators. If the input data can be directly sent to the actuator, this device (the actuator) can be specified at this moment in the column furthest to the right. The list of available devices are obtained from a repository that our tool interfaces with.

The analysis performed on the input port is supported by the second step of the *ECPSModeling* tool (Fig. 19). Analyzing the input ports of the selected UAV mathematical block, two ports are detected: (*ServosRefAngles* and *ESCsRefVelocity*). These ports contain two vectors of two entries each, representing the control references. More specifically, they provide the reference angles to the servomotors and the reference velocities to the motors. This data requires a pre-processing process before being sent to the respective actuators.

When the inputs need pre-processing, the data is supposed to be modified by the use of functions that transform the received

Figure 19 – ECPSModeling analyzing the block inputs.



information, which can then be sent to the system actuators or other functions. The design of functions related to actuation is detailed in activity 3. If for instance no pre-processing is required, this activity can be skipped.

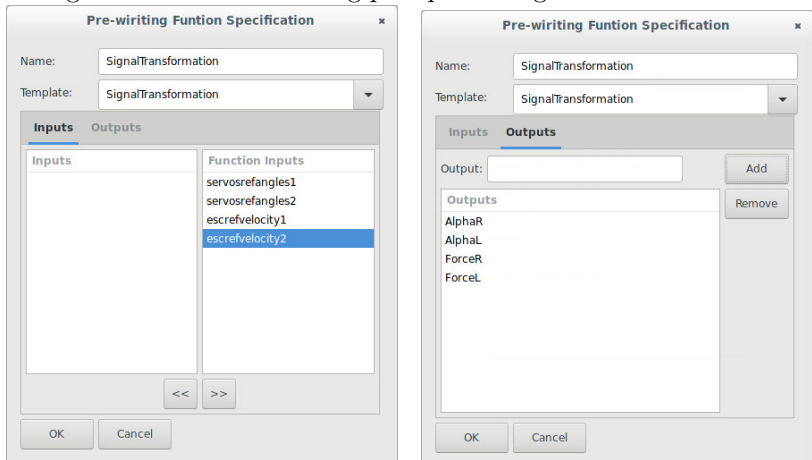
**Activity 3 Define pre-processing functions** - *The set of inputs that requires pre-processing is used as input for this activity. Here the designers must create functions that will receive the selected inputs and provide the data that will be sent to the system actuators or another functions. System functions templates can be created at this time and these structures will allow the functions to be reused in other projects.*

The UAV system is composed of a single function called *SignalTransformation()* used to perform the data pre-processing. It an object must be inserted after receives as inputs (*ServosRefAngles* and *ESCsRefVelocity*) and generates four outputs. These outputs represent the forces to the motors (*ForceR*, *ForceL*) and the angles to the servomotors (*AlphaR*, *AlphaL*). The characteristics of this function are presented in the Fig. 20.

Based on the inputs that do not require pre-preprocessing, and the outputs from the pre-processing functions, the designers need to specify the set of required actuators. The actuator's specification is detailed in activity 4.

**Activity 4 Actuator specification** - *It must be connected to an actuator every output of the pre-processing actuation functions and also the inputs from the math block that do not require pre-processing. As already mentioned, the list of actuators is obtained from a*

Figure 20 – ECPSModeling pre-processing functions definition.



repository. At this moment the designer can parameterize actuator specific information, such as priority and periodicity.

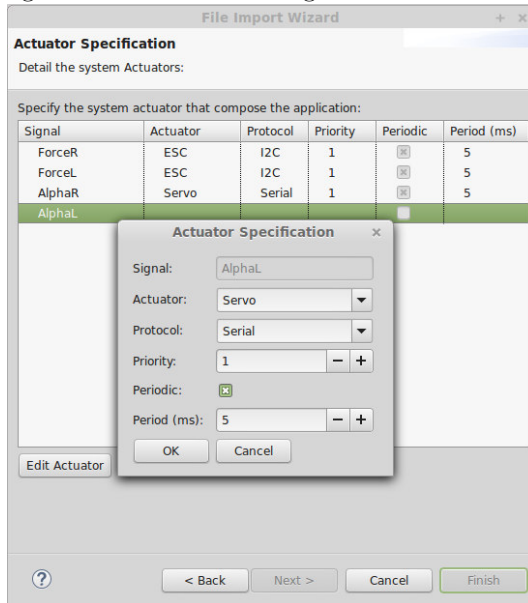
Considering the UAV actuators specification, it has been observed that four actuators are required: two motors (propellers) and two servo motors. These actuators are associated with the outputs from the previously defined *SignalTransformation()* function. The UAV actuators definition is depicted in Fig. 21.

Once the actuation functions and devices are specified, its functionalities need to be associated with software components. The refinement of the actuation software structures is suggested in the activity 5.

**Activity 5 Actuation software specification** - *This activity represents the definition of the actuation software components. Such structures describe the arrangement of the system actuators and their related pre-processing functions. Each structure can be defined as periodic or aperiodic. At this moment characteristics like activation pattern, period, priority, actuators and functions are specified for each created structure. Templates can also be created for the actuation software structures, allowing their reuse in other projects.*

The actuation software specification is supported on the *ECPSModeling* as presented in the Fig. 22. The UAV actuation subsystem is defined in a single thread called *Actuation* that performs

Figure 21 – ECPSModeling actuators definition.

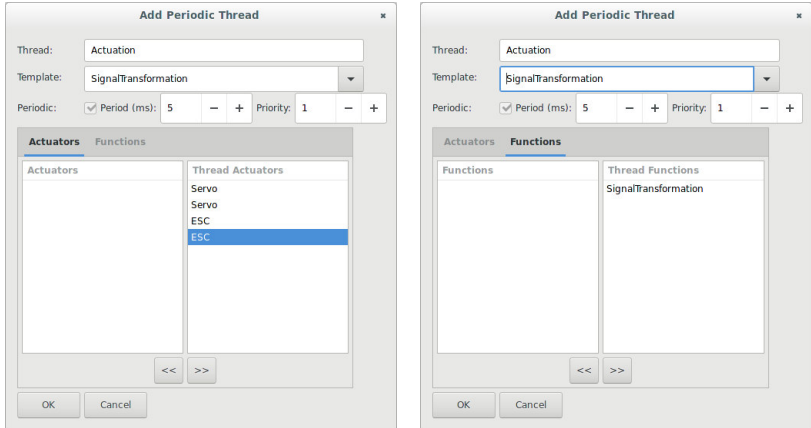


the actuation function ( $SignalTransformation()$ ) and provides the actuators interface. This structure is defined as periodic, with a 5 millisecond period.

Activity 5 ends the first part of the workflow (the green blocks presented in Fig. 17). The next part of the workflow is quite similar, but works in the opposite direction, i.e., analyzing the outputs of the math component block. Activity 6 starts this part of the workflow, where the designers analyze the outputs of the selected block, as further described.

**Activity 6 Output analysis** - Here the outputs of the selected mathematical block are presented to the designer, who needs to define characteristics such as the type and size of the data, whether post-reading is required or not, and the related sensors (from the repository). Similarly to actuators, the sensor can only be immediately selected if no post-reading is required. Details related to the Post-reading option, which means that the selected output data cannot be directly obtained from a specific sensor, thus requiring some kind of post-reading processing to transform the sensor data into the system information (by the filter application for example) in order to estimate the system

Figure 22 – Define the actuation software structure.



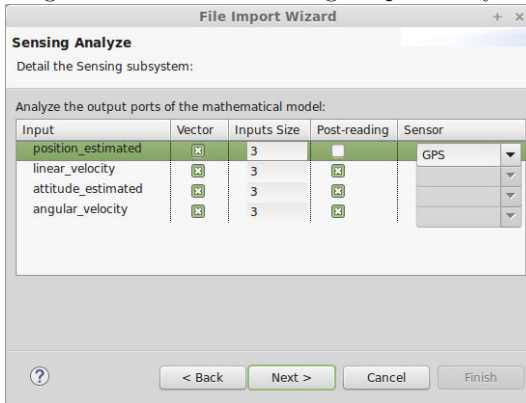
behavior. If the output can be directly obtained from a specific sensor, this device (the sensor) can be specified at this moment in the most right column of the form. The list of available sensors are obtained from a repository that maintains the devices characteristics.

The proposed output analysis is supported by *ECPSModeling* tool, as can be seen in the Fig. 23. Analyzing the outputs of the selected mathematical block (see the *UAV\_Dynamics* block outputs from Fig. 38), it is observed that the UAV system has four outputs that represent, respectively, the position (*position\_estimated*), the attitude (*attitude\_estimated*), and the linear and angular velocities (*Linear\_velocity* and *Angular\_velocity*). These outputs contain four vectors with three elements each and cannot be directly obtained by a specific sensor. In this way, these outputs are defined as requiring a post-reading processing to estimate the system behavior.

When the outputs need a post-reading processing, the data from the respective sensor is supposed to be modified by the use of functions that transform it. Normally the system states cannot be directly obtained by a unique sensor, and the data from different sensors needs to be fused in order to provide better accuracy on the estimated behavior.

The design of functions related to sensing is detailed in activity 7. If for instance no pre-processing is required, this activity can be skipped.

Figure 23 – ECPSModeling output analysis.



**Activity 7 Define post-reading functions** - The set of outputs that requires a post-reading processing are used as input of this activity. Here the designers must create functions that will receive the sensor data and forward the data to another function. System function templates can be created at this time, allowing its reuse in different projects. After performing the functions design, the designers define the required structures to perform the behavior estimation.

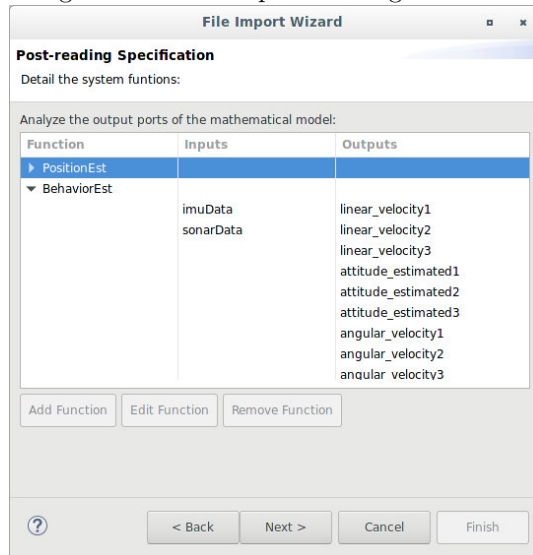
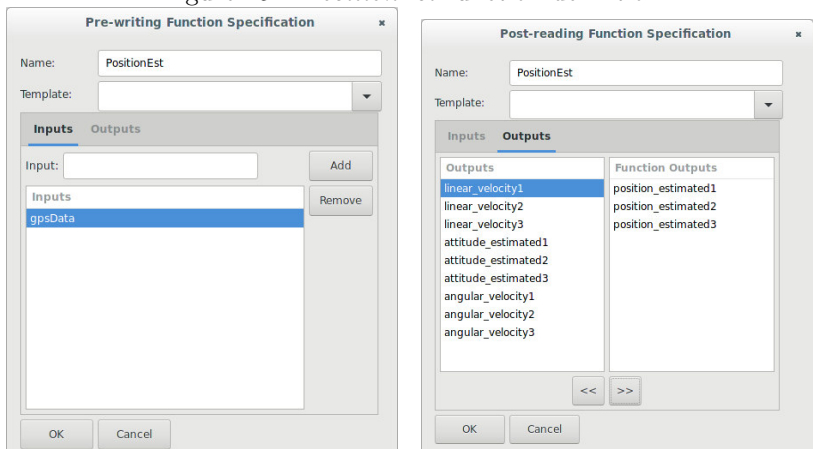
In the UAV project, given that sensors with different sampling periods are used, two sensing functions were created. These functions are depicted in the Fig. 24. One is used to perform the interface with the sensors that have low sampling periods (*BehaviorEst()*) and another to support the sensors that have higher sampling periods (*PositionEst()*).

The *BehaviorEst()* function provides the sensor interface and executes a complementary filter, responsible for fusing the data from the different sensors, while *PositionEst()* provides the interface with the GPS device. The information required to design the *PositionEst* function is presented in Fig. 25

Based on the outputs that do not require post-reading, and on the inputs of the sensing functions, the designers need to specify the set system sensors. The definition of these components is based on the sensor's characteristics and the project requirements. The specification of sensors is detailed in activity 8.

**Activity 8 Sensor specification** - Each input of the post-reading

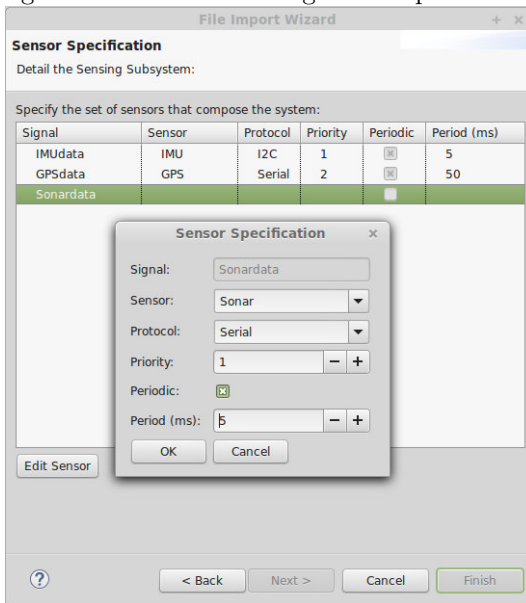
Figure 24 – Define post-reading functions.

Figure 25 – *PositionEst* function definition.

functions and the outputs which no post-reading processing is required are connected to a sensor. In fact, this activity aims to arrange the described signals with a set of required components in order to provide the behavior estimation. The available sensors are obtained from a repository. The sensors also need to be characterized.

As can be seen in the Fig. 26, the UAV system is composed of three sensors. These sensors send their data to the sensing functions that are responsible for fusing the data and providing the UAV behavior estimation. The required sensors are: (i) an IMU, (ii) a Sonar, and (iii) a GPS. The IMU and the sonar have smallest sampling period compared to the GPS, with values of 5 milliseconds and 50 milliseconds respectively.

Figure 26 – ECPSModeling sensor specification.



Once the sensing devices and functions have been specified, these components also need to be organized in software structures providing the behavior estimation. The definition of these structures is detailed in the activity 9.

**Activity 9 Sensing software specification** - This activity details the definition of the sensing software components. Such structures



Figure 27 – ECPSModeling sensing threads specification.

**File Import Wizard**

**Sensing Threads Specification**

Organize the system threads that manage the system sensors:

Periodic Threads:

Thread	Period (ms)	Priority	Sensors	Functions
BehaviorEstimation	5	1		
			IMU Sonar	BehaviorEst
PositionEstimation	50	2		
			GPS	PositionEst

Sporadic Threads:

Thread	Period (ms)	Priority	Sensors	Functions

*describe the arrangements of the system sensors and their related post-reading functions. Each structure can be defined as periodic or aperiodic. At this moment characteristics like activation pattern, period, priority, actuators and functions are specified for each created structure. Templates can also be created for the actuation software structures, allowing their reuse in other projects.*

Based on the system sensors and the sensing functions, the software structure of the UAV sensing subsystem is defined. As shown in Fig. 27 two threads were created to support this subsystem.

The first called *BehaviorEstimation* is responsible for interfacing with the IMU and sonar, and for performing the complementary filter. It has a period of 5 milliseconds. The second thread, (*PositionEstimation*), is responsible for interfacing with the GPS and provides the positional information. It is periodic and has a period of 50 milliseconds.

After having defined the sensing and actuation subsystems characteristics, the model transformation can be performed, generating the output model representation. This transformation relates to activity 10.

**Activity 10 Model transformation** - Based on the original Simulink model, which now is associated with the specified sensing and actuation components, the model transformation process is performed generating the AADL architectural model.

By the use of proposed activities supported by the ECPSModeling tool, the design process of sensing and actuation subsystems is systematized, ensuring that the designers will cover the required steps to specify the set of system devices. In this way, the proposed approach provides the means to represent both the set of system sensors and actuators as well as to represent the required software structure to perform the system actuation and behavior estimation.

In order to help better understand the application of the mentioned activities and the proposed tool, a video was created to illustrate the complete process (see <https://youtu.be/hBwQ2tfLx1I>). The next subsection details the characteristics of the AADL model that is generated after using the proposed tool.

### 5.2.3 Output Model Generated by the Tool

By the use of *ECPSModeling* the AADL architectural model is generated. This model represents the integration between hardware and software components, including the cooperation between the system devices, processes, and threads. The initial structure of the generated AADL model from the UAV case study is presented in Fig. 28. This structure details the components of the UAV architecture.

As can be observed in lines 4 and 5 of the AADL model, the system architecture is composed of two processes: *pi\_control\_system*, and *pi\_sen\_act*. The model also contains seven devices (lines 7 to 13) (*di\_escR*, *di\_escL*, *di\_servoR*, *di\_servoL*, *di\_gps*, *di\_sonar*, and *di\_imu*). In addition, lines 15 to 35 are used to declare the system connections.

The control approach is composed of one process, which is responsible for performing the system stabilization and the path tracking controls. The sensing and actuation subsystems are managed by the *pi\_sen\_act* process. This representation provides the sensor and actuator interface and supports the execution of the estimation algorithms. The AADL structure that details the sensing and actuation process is presented in Fig. 39.

The *p\_sen\_act* process is composed of three threads (*ti\_behaviorEst*, *ti\_positionEst* and *ti\_actuation*). The *ti\_behaviorEst*

Figure 28 – UAV model with sensing and actuation process.

```

1  SYSTEM IMPLEMENTATION UAV.impl
2  SUBCOMPONENTS
3    --PROCESS
4    pi_control: PROCESS p_control.impl;
5    pi_sen_act: PROCESS p_sen_act.impl;
6    --DEVICE
7    di_esc_r: DEVICE d_esc.impl;
8    di_esc_l: DEVICE d_esc.impl;
9    di_servo_r: DEVICE d_servo.impl;
10   di_servo_l: DEVICE d_servo.impl;
11   di_gps: DEVICE d_gps.impl;
12   di_sonar: DEVICE d_sonar.impl;
13   di_imu: DEVICE d_imu.impl;
14   CONNECTIONS
15   C1: PORT di_gps.position -> pi_sen_act.position;
... Here goes all the connections (lines 16 to 35)
35 END UAV.impl;

```

Figure 29 – AADL representation of sensing and actuation process.

```

1  PROCESS IMPLEMENTATION p_est_act.impl
2  SUBCOMPONENTS
3    ti_behaviorEst: THREAD t_behaviorEst.impl;
4    ti_positionEst: THREAD t_positionEst.impl;
5    ti_actuation: THREAD t_actuation.impl;
6  CONNECTIONS
7    C1: PORT distance -> ti_behaviorEst.distance;
... Here goes all the connections (lines 8 to 25)
26 END p_est_act.impl;

```

and *ti\_positionEst* are responsible for the sensor's interface and the execution of the estimation algorithms. These threads fuse the received data and provide the system behavior estimation. The system actuation is provided by another thread (*ti\_actuation*) that supports the system actuation, i.e., the software interface to a set of actuators in order to manage the process under control according to the control references.

### 5.3 SUMMARY

As presented the *ECPSModeling* tool was created in support for the automated design of CPS, especially in the definition of the sensing and actuation subsystems. The tool supports the application of the

design activities promoted by the authors in order to reorganize the functional model or, more specifically, the mathematical model used for simulation purposes. In other words, it allows systematizing the process of designing the sensing and actuation subsystems.

By the use of the *ECPSModeling* tool, designers can properly create the required sensing and actuation subsystems in the generated AADL model and, consequently, in the final application. The generated AADL model integrates the control algorithms, estimation filters, and the devices (sensors and actuators) used in the final application.

The proposed tool does not intend to provide specific solutions to solve particular details related to the design of the sensing and actuation subsystems. In fact, it guides designers while performing necessary design decisions, making the process less prone to errors. In addition, it allows for the reuse of previously defined code for sensors and actuators, including estimation filters and device drivers. As previously mentioned, an authors' decision for starting the process with the actuation subsystem does not represent a strict design decision. There would be no problem starting the process with the sensing subsystem instead.

Besides the automation of the sensors and actuators integration, supported by the *ECPSModeling* tool, another activity proposed in the CPS design method described in the Chapter 4 was automated by the tool's usage. This activity is related to the integration of formal verification methods, more specifically MC in the design method. Details related to the proposed integration are described in the next chapter.

## 6 INTEGRATING FORMAL VERIFICATION INTO THE UAV DESIGN

CPS's are typically complex systems and their design process requires strong guarantees that the specified functional and non-functional properties are satisfied for the designed application. Adequate tools and methods are of utmost importance to support and guide the teams in order to increase the potential of the project success (DERLER; LEE; VINCENTELLI, 2012). To ensure requirement fulfillment along the design process, strong system analysis is needed (HUTH; RYAN, 2004).

Considering the typical high complexity of UAV systems and the need for a strong analysis to ensure the design correctness, formal verification becomes a natural candidate to become part of the overall CPS design process (YU et al., 2011). Different approaches exist, for example Model Checking (MC), Theorem Proving, and Runtime Verification (RV). Each method has its pros and cons, namely: MC suffers from the state explosion problem; Theorem Proving requires highly technical knowledge and despite its latest developments it still faces many automation problems (due to foundational limitations on the supporting logical theories); and RV brings overhead to the CPS since monitors have to be coupled with system components and process extra information from events.

Aiming to avoid the state space explosion in Model Checking, different design techniques can be applied. Examples of these techniques include: abstractions and reduction of unnecessary states; the use of symbolic model checking, applying binary decision diagrams and symbolic algorithms; the partial order reduction that concerns the identification of interleaving sequences, eliminating redundancy thus reducing the state space.

Given this scenario, MC seems to be the more natural approach for our work since it conforms to MDE practices and can be fully automated. Well-known MC tools are UPPAAL (BEHRMANN; DAVID; LARSEN, 2004b), HyTech (HENZINGER; HO; WONG-TOI, 1995), Kronos (BOZGA et al., 1997), among others. Given that UPPAAL performance is much better than other tools such as HyTech, and Kronos in time and space (BENGTSSON et al., 1995). In this context, on the present work we adopt MC by using the UPPAAL tool (BEHRMANN; DAVID; LARSEN, 2004b).

However, when looking at the existing MDE tool-support, it is

observed that there is still little support for the automated integration of formal verification techniques in these tools. Given that formal verification is necessary to ensure the levels of reliability required by safety critical UAV. The integration of formal verification methods on the UAV design process is also covered by the method proposed in the Chapter 4. In the proposed method it was represented in the UAV detailed project (Step 2.4), the formal verification process (Activity 2.4.3) that regards the evaluation and validation of the system properties by using formal methods.

In order to support the formal verification based on MC, a timed automata should be created to express the system behavior, supporting the evaluation of different properties such as safety, reachability, liveness, and deadlock. However, generating these representations is not a simple task and requires sufficient knowledge of the design team to correctly express the system properties. To automate the timed automata construction, it is our claim that a model transformation can be performed using as input the architectural representation.

In this chapter an approach to applying an MC technique to the UAV design is made, allowing the timing and error properties evaluation to be presented. The proposed approach includes a model transformation process that based on the architectural model in AADL (FEILER; GLUCH; HUDAK, 2006), generates a Timed Automata representation that conforms the UPPAAL tool (BEHRMANN; DAVID; LARSEN, 2004b). The tool named *ECPS Verifier* was created in the context of this proposal to support such model transformation.

The proposed transformation process can be defined as a specialization of the UAV design method described in Chapter 4. More specifically the proposed approach describes a set of activities aiming to guide the designers to extract the system behavior (represented by timed automatas), based on architectural representation (Activity 2.4.3 of the proposed method). The proposed method is illustrated by means of the design of a UAV, from where we derive the timed automata models to be analyzed in the UPPAAL tool.

## 6.1 FORMAL VERIFICATION OF AADL ARCHITECTURAL MODELS

Aiming to provide the evaluation of the CPS properties by using the MC a design method was created to help designers to integrate this technique into the UAV design process. In this sense, different

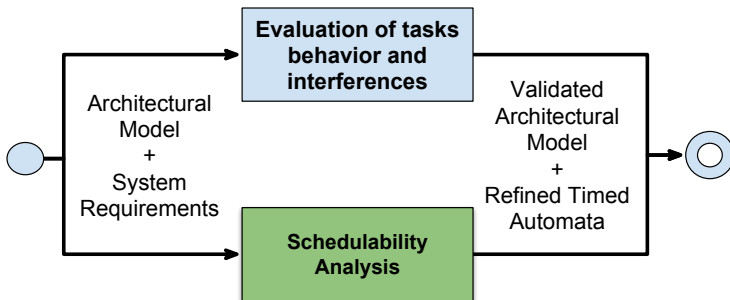
properties can be evaluated from the system under designs such as liveness, reachability, deadlock freeness, and others. To support this integration a tool named *ECPS Verifier* was designed to provide the model construction.

*ECPS Verifier* consists of the tool support to allow performing a model transformation, moving from an architectural model in AADL to a network of timed automata devoted to be analyzed via MC. The use of AADL components designed in the OSATE tool is assumed to represent the architectural model and the use of timed automata suited to the UPPAAL tool to represent the system behavior.

An important aspect to be highlighted is that the problem under consideration is not restricted to simply representing the UAV behavior and analyzing its properties. It happens that the designer must properly plan how to express the system properties according to the proposed architecture and the set of system devices, evaluating how correctly it expresses the system properties in order to provide guarantees that the system fulfills its restrictions.

A set of design activities are proposed to integrate MC into the UAV design process, allowing the formal system verification according to predefined properties and restrictions. The overall process is composed of two main phases, the *evaluation of tasks behavior and interferences* and the *schedulability analysis* (see Fig. 30). These phases are proposed to be performed at independent moments, providing a means for the design team to work in tandem to evaluate different system characteristics, ensuring property validation.

Figure 30 – *ECPS Verifier* Top View.

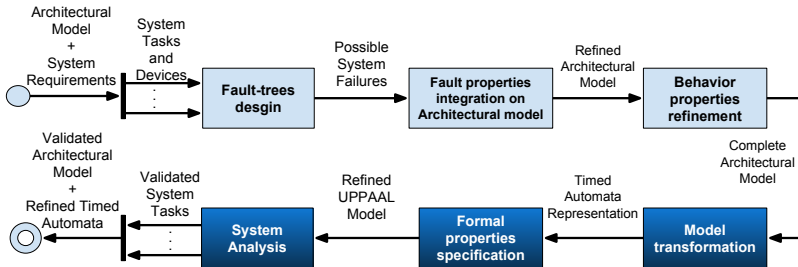


### 6.1.1 Phase 1: Evaluation of Tasks and Interferences

In this phase the designer individually analyzes the set of UAV task characteristics, in order to validate the relationship between the system tasks and the associated set of devices. At this time, the interference caused by the other tasks is not considered. As the output of this phase, a set of timed automatas and a refined AADL architectural model are provided.

The first proposed phase defines a sequence of activities applied to each defined task, in order to allow its formal property evaluation. These activities are presented in Fig. 31, aiming to properly guide the design teams to represent the task behavior and the possible device faults. This information is integrated into the architectural model (by the use of AADL annexes) to support the MC evaluation.

Figure 31 – ECPS Verifier Tasks Behavior and Interferences evaluation.



This phase receives the AADL architectural model as input coupled with the set of system requirements. To provide the system analysis each system task is isolated and its behavior evaluated individually, by performing the proposed activities.

**Activity 1 Definition of fault-trees** - *In this activity, the designer analyzes the UAV task characteristics aiming to specify the possible failure events. This information includes the proposed mission/task for the UAV, its configuration, the set of required devices, and restrictions. The output of this activity is a fault-tree to be used by the designers.*

Based on the fault-tree it we are able to evaluate the implications of the related events to the task behavior and define alternatives to mitigate their effects. These representations aims to provide a top



view of the possible system failures that should be considered during the design process.

**Activity 2 *Integration of fault properties in the architectural model*** - *By using the AADL Error Annex (EA) (SAE, 2011a) the fault-tree is added to the architectural model, representing the possible failures, and the associated probability for each error event occurrence. This information is used to evaluate the failure impact in the designed system, as well as to define alternatives to mitigate their effects. The output of this activity is a refined architectural model, integrating the error properties.*

To provide the automata generation the defined behavioral UAV properties needs to be refined.

**Activity 3 *Refinement of behavior properties in the architectural model*** - *The behavioral properties are refined in this activity. In this way, based on the system characteristics, and by using the AADL threads and the AADL Behavioral Annex (BA) (SAE, 2011b) properties, the system characteristics are specified as execution states, system variables, system transitions, and subprograms access. The output of this activity is a refined architectural model, that integrates error and behavioral properties.*

Regarding the usual AADL model design, it is observed that these models typically contain behavioral information that allow designers to evaluate system properties. However, to the timed automata extraction, some properties need to be refined, adding information related to the automata guard and variable declaration. Such complete AADL model is suitable to be submitted to the transformation process.

**Activity 4 *Formal verification*** - *In this activity the designers perform the system evaluation based on the generated architectural model. This activity is split into three sub-activities (three last blocks from Fig. 31), that address each part of the formal verification process.*

The verification process is based on the timed automata (UPPAAL model) that is generated by performing a model transformation, whose input is an AADL model.

**Subactivity 4.1 *Model transformation*** - *A UPPAAL model is created for each designed task by means of a model transformation*

*process from the AADL specification. Such a UPPAAL model is composed of a set of templates that describe the AADL task behavior and device characteristics. The transformation is based on a set of rules that map the source and target models, as detailed in Section 6.2.*

**Subactivity 4.2 Formal properties specification** - *In this activity, the designers create formal expressions that represent model properties that must be evaluated. These properties represent the characteristics and restrictions that the system needs to meet to fulfill its objective. In this sense, properties like liveness, reachability, safety, and deadlock occurrence can be evaluated.*

However, formally representing the system properties - in this case using the UPPAAL syntax (and the UPPAAL-SMC syntax for those properties that have probabilities associated) - is not a simple task. This comes from the fact that these expressions are dependent on the adopted formal language and, in addition, describing the system restrictions using a formal language is not trivial and requires considerable knowledge.

**Subactivity 4.3 System analysis** - *The system analysis is performed by checking the validity of the formally specified properties carried out in Activity 4.2, against the models derived in Activity 4.1. Depending on the results, system changes may be required (which implies regenerating of, at least, some of the existing automata), therefore adjusting the system to satisfy its intended behaviors.*

Performing the property evaluation it is possible to observe whether or not the system meets its requirements. This implies that the safety properties are satisfied and that no deadlocks are found except if an error occurs. The performed analysis on the UAV system is detailed in Section 6.4.

Coupled with the task behavior evaluation, the system schedulability needs to be validated. This validation process is proposed on the second phase (*Schedulability Analysis*), that proposes a set of activities to validate these properties.

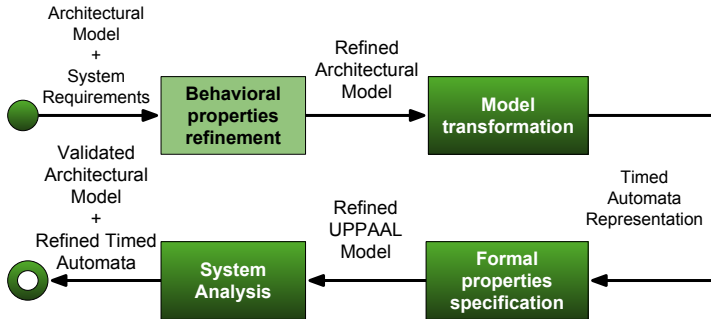
### 6.1.2 Phase 2: Schedulability Analysis

The schedulability analysis relates to the evaluation and validation of the set of system tasks according to a defined scheduling police. At this time, characteristics such as tasks interferences, and

defined priorities are considered to validate system characteristics. The set of schedulable tasks and a refined architectural model is provided as output.

This phase defines a sequence of activities that is applied to the set of tasks validating its integration and the system execution. These activities are presented in Fig. 32, that aims to properly guide the design teams to evaluate the tasks set properties, integrating this information into the architectural model to support the MC evaluation.

Figure 32 – ECPS Verifier Schedulability Analysis.



**Activity 1 Behavioral Properties Refinement** - The behavioral properties refinement regards the improvement of the UAV system characteristics, including information such as tasks periods, priorities, execution time among others. Based on the AADL threads and the BA properties the designers detail the task characteristics that will support the schedulability analysis.

Once the tasks temporal characteristics are refined, the system is able to be submitted for formal verification. Regarding the scheduling police, as will be described in Section 6.2.3, at this time only the Rate-Monotonic scheduling algorithm is implemented.

**Activity 2 Formal verification** - In this activity the designers perform the system evaluation based on the generated architectural model. This activity is split into three sub-activities (three last blocks from Fig. 32), that address each part of the formal verification process.

The verification process is based on the timed automata (UPPAAL model) that is generated by performing a model transformation, whose input is an AADL model.

**Subactivity 2.1 Model transformation** - *An UPPAAL model is created to represent the set of system tasks by means of a model transformation process from the AADL specification. Such a UPPAAL model is composed of a set of templates that describe the AADL task characteristics. In this evaluation process the set of system devices are not directly considered, and its interferences characteristics are integrated into the task models. The transformation is based on a set of rules that map the source and target models, as further detailed in Section 6.2.*

**Subactivity 2.2 Formal properties specification** - *In this activity, the designers create formal expressions that represent model properties that must be evaluated. These properties represent the characteristics and restrictions that the system needs to meet in order to fulfill its objective. In this sense properties such as liveness, reachability, safety, and deadlock occurrence can be evaluated.*

**Subactivity 2.3 System analysis** - *The system analysis is performed by checking the validity of the properties specified in Activity 4.2 against the models derived in Activity 4.1. Depending on the results, system changes may be required (which implies regenerating of, at least, some of the existing automata), therefore adjusting the system to satisfy its intended behavior.*

Applying the schedulability analysis designers can evaluate not only if the tasks meet their deadlines or not, but also if the defined priorities are adequate and if no deadlocks are found - except in the event of errors occurring.

### 6.1.3 Final Remarks

As described in this section the proposed formal verification of AADL architectural models is composed by two phases. They include, respectively, the tasks behavior evaluation and the schedulability analysis. By performing the set of proposed activities different properties are evaluated and validated, ensuring the analysis of characteristics such as reachability, safety, liveness, deadlock freeness, schedulability, among others.

The first phase (*Evaluation of tasks behavior and interferences*) supports the task characteristics analysis, i.e. evaluating each task individually. This phase is started with an activity that targets

analyzing the possible UAV faults (*Fault-trees design*), having as output a fault-tree definition. Based on the defined faults, a refinement is performed in the architectural model (*Fault properties integration on architectural model*) by integrating the fault's characteristics. This activity is supported by using the AADL EA. In addition, the threads behavior is also evaluated, refining its properties by using the AADL BA (*Behavior properties refinement*). The refined architectural model is used as input to the model transformation process (*Model transformation*), performed by the *ECPS Verifier* tool, generating the UPPAAL automatas. Then the system requirements are encoded as temporal formulas in the UPPAAL syntax (*Formal properties specification*), generating a complete UPPAAL model. Finally, the system is evaluated (*System analysis*).

On the other hand, the *Schedulability analysis* aims to validate the execution of the set of proposed tasks. The method starts with an activity that targets the refinement of the threads behavior properties on the AADL architectural model, by using the AADL BA (*Behavioral properties refinement*). The refined architectural model is used as input to the model transformation process (*Model transformation*), performed by the *ECPS Verifier* tool, generating the UPPAAL automatas. Then the scheduling system requirements are encoded as temporal formulas in the UPPAAL syntax (*Formal properties specification*), generating a refined UPPAAL model. Finally, the system schedulability is evaluated (*System analysis*), providing the validated system timed automata representation and the refined architectural model.

The proposed phases were designed to be performed either in parallel or sequentially. If any characteristic were to be adjusted in a model, the generated subsystems need to be updated in order to provide the complete application evaluation and validation.

To support the related design activities, especially the transformation process, the *ECPS Verifier* tool was designed. It receives the AADL model as input and performs a transformational process to provide the generation of the timed automatas representation. A detailed description of this tool is presented in the following section.

## 6.2 MODEL TRANSFORMATION TOOL *ECPS VERIFIER*

To automate the UPPAAL model (automata) generation from the AADL model, and to make this process less error prone, we have developed a tool named *ECPS Verifier*. This tool follows the MDE principles, which state that a mapping between the source (AADL) and the target (UPPAAL) model is defined by means of transformation rules. Auxiliary structures are required to provide the automata execution management. The transformation rules make use of metamodels from both source and target models, as further detailed.

### 6.2.1 Related Metamodels

The AADL (source) metamodel is composed of a root *System* that contains a set of subcomponents representing the other AADL components, such as *Processes*, *Threads*, *Ports*, and *Connections*. It is important to highlight that the *Thread* component encapsulates the system behavior, so it can contain subcomponents that may represent software calls (black-block component) or it can detail the behavior by means of state machines using the BA. Given that our approach is more focused on the AADL software components, the single hardware element under consideration is the *Device*. A representation of AADL meta-model is presented in Fig. 33.

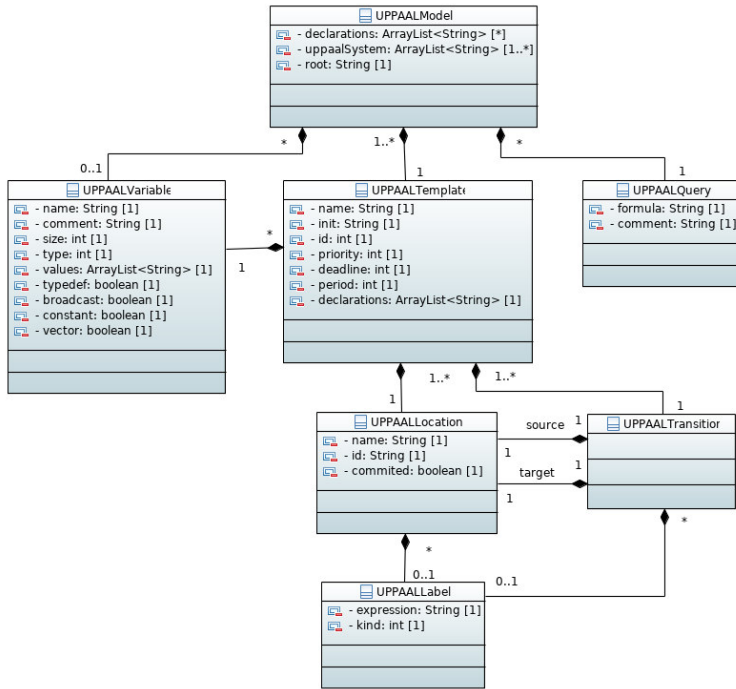
Overall, the target metamodel is composed of a set of templates that encapsulate the timed automata, and by a set of queries used for the model evaluation. UPPAAL models are composed of at least one template containing a timed automata, which is decomposed into a set of states and transitions. These transitions can incorporate restrictions (guards and synchronizations points), and can also incorporate actions expressed as in an imperative programming language - it allows us to declare variables and make function calls. Such actions are defined in the update definitions. Finally, a set of queries describing the properties to be evaluated can also be attached to the model, this meta-model is presented in the Fig. 34.

### 6.2.2 Transformation Process

To perform the transformation process, both a parser and a transformation engine were created, all developed in Java as a plug-in



Figure 34 – UPPAAL meta-model.



from Osate tool. The parser is responsible for mapping the source (AADL) textual model to memory in accordance with its related metamodel elements.

When the transformation process is started, a parser is applied to the selected AADL input file. This parser is composed of a set of rules that identify the AADL structures and create Java objects mapping the system characteristics, such as process organization, threads information (period, deadline, execution time, behavior), devices characteristics (priority, execution time, error properties), among others. The set of AADL ports and connections are also represented, describing the relationship between the system components and their information exchange.

The transformation engine is responsible for performing the automata generation based on the AADL source model. This engine is composed of the following rules:



- AADL *Thread* components are mapped to UPPAAL templates representing its behavior and characteristic like states, guards, invariants, periodicity, priority and others.
- AADL *Devices* are also mapped as UPPAAL templates, detailing its behavior coupled with its possible failures.
- The set of input and output ports from the AADL model are used as basis to the variables and UPPAAL channels declarations, representing the system communication.

The devices model are composed essentially by a set of five states, the *idle* state and the *execution* state (Sensing or Actuate) represent the regular operation of these systems. Three other states define the atypical operation of this component, representing its *partial operation*, the system *emergency mode*, and an *irreversible failure* occurrence. Attached with this set of states are the transitions, which aim to express the system behavior. Probabilities are associated to each error state, representing the occurrence distribution of the considered errors.

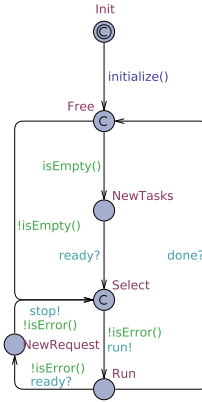
To provide more complete automata representations auxiliary components are required coupled with the threads and devices templates. In this sense, UPPAAL templates are proposed, as well as some design conventions, in order to details the model representation.

### 6.2.3 Auxiliary Components

To manage the UPPAAL templates execution a scheduler structure was proposed, providing the tasks activations according to their periods. Currently, only a Rate-Monotonic (RM) (LIU; LAYLAND, 1973) scheduling algorithm - applied to a single-core processor - is used to provide the tasks activation according to their priorities. The scheduler model is presented in Fig. 35.

This scheduler is composed by an initial function (*initialize*) that performs the threads inclusion on the scheduling queue, according its priorities going to the *Free* state. Then the thread with the highest priority is selected (*Select*) and its execution is started (*Run*). If a new task is available during a task execution (*NewRequest*), the scheduler stops the active thread and evaluate the priorities of the running and the new activated thread. If the running task has the highest priority it is resumed, but if the new task has a higher priority, the running thread is blocked and the new thread is executed.

Figure 35 – Scheduler model.

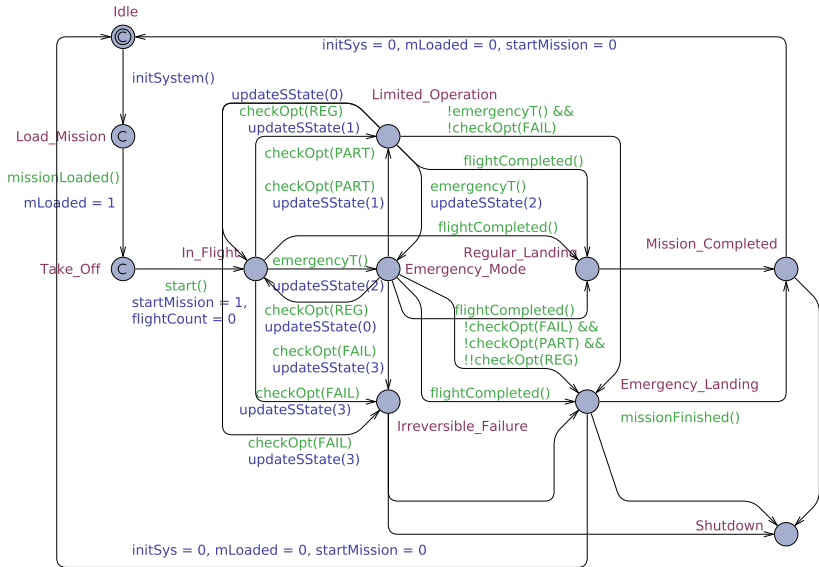


The *ECPS Verifier* scheduler model was designed to be automatically included into the UPPAAL model along the transformation process. Thereby, designers only need to include a device named *scheduler* in the AADL model.

Regarding the UAV behavior and its set of operational modes, a behavioral representation was proposed aiming to detail the system operation. It is our concern that the defined behavior can be described as some kind of generalization, in order to represent a typical UAV operation. This representation is illustrated in Fig. 36.

The UAV behavior model is composed of an initial model (*Idle*) that is activated when the system is turned on. Then an function (*initSystem()*) is invoked to ensure the system initialization. At this state (*Load\_Mission*), the UAV waits for a new mission, flagged by the *missionLoaded()* function. As the mission is loaded, the UAV takes off and starts to perform the mission (*In\_Flight*). During the flight four different scenarios are defined. The first scenario is the regular UAV operation, when the UAV finishes the proposed flight (*flightCompleted()*), it performs a regular landing (*Regular\_Landing*), and ends the proposed mission (*Mission\_Completed*). Completed, the mission the UAV can be turned off (*Shutdown*), or the system can be put into standby waiting for a new mission (*Idle*). The second scenario describes the activation of the limited operation during the mission execution (*Limited\_Operation*), this can be caused by some unusual behavior or some system components (devices), in this way, according the failure level the UAV can perform a regular or emergency landing

Figure 36 – UAV Behavior model.



(*Regular\_Landing* or *Emergency\_Landing*), if the regular landing is performed the UAV finishes its operation as described in the first scenario. However if the emergency landing is performed, according the UAV damage level and the mission status, the UAV can signal the ended mission (*Mission\_Completed*) or shutdown the system (*Shutdown*). According the UAV damage level an emergency mode can be activated (third scenario), in this way, the UAV performs an emergency landing (*Emergency\_Landing*) and, according to its damage level, the system can be turned off, or the ended mission flagged. Finally if irreversible damage is identified, the system activates the irreversible failure mode (*Irreversible\_Failure*), where according to its damage level the system can be turned off or an emergency landing can be performed.

The UAV behavior model was designed to be automatically included during the transformation process on the *ECPS Verifier*. Thereby, designers only need to include a device named *behavior* in the target AADL model.

Regarding the AADL model design, in order to support the required UPPAAL properties representation in the AADL model some design conventions were established, ensuring the mapping between

UPPAL properties and the AADL model. These characteristics require the AADL subprograms declaration to represent the UPPAAL invariants, the rate expressions, the transition guards, and functions of the designed scheduling approach, as well as data types are defined to represent the UPPAAL channels (*chan*) and the clock variables (*clock*).

In Fig. 37, from lines 2 to 5, the structure of the designed AADL subprogram to represent the UPPAAL location invariants is detailed. This subprogram provides the representation of the states invariant defined by its name and the invariant expression. A subprogram that details the UPPAAL transition guards is presented in lines 7 to 11. This subprogram is needed due to the fact that guards of the AADL BA do not support the use of functions in the guards expression. Thereby, this subprogram is composed of three inputs describing the UPPAAL function that should be declared as a subprogram, the guard operator, and the value used in the expression. Another subprogram structure that details the UPPAAL initial values definition is defined from lines 13 to 16. This subprogram supports the definition of initial values to the created variables, this functionality is not natively supported on AADL. This subprogram is composed of the variable name (*var*) and its initial *value*.

As described, a scheduler model can be added to the UPPAAL model, performing an RM algorithm. When inserted, this model includes a set of functions that describes: the addition of a new task in the scheduler queue (Fig. 37, lines 19 to 21); removal of the task from the top of the queue, i.e., the running task (Fig. 37, lines 231to 2419 a function that returns the running task (Fig. 37, lines 261 to272).

Two data types are declared in the AADL model. These data types represent the UPPAL channels (Fig. 37, line 30), and by convention all channels are defined as broadcast, and the *clock* type (Fig. 37, line 32) describes the UPPAAL clock variables.

Due to the fact that the AADL model can be composed of multiple systems and implementations, in our approach we define that the first translated system is defined as the root of the translated system. In this sense the designer needs to first declare the root system in the AADL file. Once the transformation is ended, the system is able to be formally verified.

Although AADL allows the system design using multiple files, on our approach the use of multiple files is not supported, the architectural model needs to be coded in one single file.

It is important to highlight that besides some design conventions being adopted, in order to provide model mapping, its verified that

Figure 37 – AADL design conventions.

```

1 -- ** SUBPROGRAMS **
2 SUBPROGRAM sp_invariant
3   FEATURES
4     state: IN PARAMETER String; inv: IN PARAMETER String;
5 END sp_invariant;
6
7 SUBPROGRAM sp_guard
8   FEATURES
9     function: in PARAMETER String; operator: in parameter String;
10    value: in parameter String;
11 END sp_guard;
12
13 SUBPROGRAM sp_init
14   FEATURES var: IN PARAMETER String;
15   value: IN PARAMETER String;
16 END sp_init;
17
18 -- ** SCHEDULER FUNCTION **
19 SUBPROGRAM sp_add
20   FEATURES id: IN PARAMETER;
21 END sp_add;
22
23 SUBPROGRAM sp_remove
24 END sp_remove;
25
26 SUBPROGRAM sp_head
27 END sp_head;
28
29 -- ** DATA TYPES ***
30 DATA chan END chan;
31
32 DATA clock EXTENDS Base_Types::Integer END clock;

```

the AADL model supports the representation of required structures to provide automata generation. In this sense, regarding that during the architectural model construction the threads behavior is specified using the BA, and that the model has a high information refinement, its possible to say that only the design conventions needs to be included on the AADL base model to enable the transformation process.

The application of the proposed activities is illustrated by means of the design of a UAV control system. More specifically, such system are devoted to representing the sensing and actuation subsystems of the UAV. The details of these activities are presented in the following section.

### 6.3 DESIGN OF SENSING AND ACTUATION SUBSYSTEMS OF AN UAV

The application of the set of activities presented in the previous section is illustrated here by means of the design - and verification - of the sensing and actuation subsystems of an UAV. Such a UAV was conceived in a related project named ProVant<sup>1</sup>, whose goal is the construction of an autonomous UAV. It tackled the design of the electro-mechanical components and the embedded control system of the UAV, both at the hardware and software levels.

Overall, the aircraft design is conducted by the use of the design method proposed in Chapter 4. Such design comprised the creation of a Simulink functional model for the control system, and of an AADL architectural model to represent the UAV embedded system. The designed architectural model is used as a basis for the approach presented in this chapter.

The top-level view of the AADL model for the UAV system is depicted in Fig. 38. This model integrates the control system (process *pi\_control\_system*, line 4) with the sensing and actuation subsystems (process *pi\_est\_act*, line 5). Coupled with these processes are the set of system devices (lines 7 to 13) that represent the required UAV sensors and actuators.

Figure 38 – UAV model with sensing and actuation process.

```

1  SYSTEM IMPLEMENTATION UAV.impl
2  SUBCOMPONENTS
3  --PROCESS
4  pi_control: PROCESS p_control.impl;
5  pi_sen_act: PROCESS p_sen_act.impl;
6  --DEVICE
7  di_esc_r: DEVICE d_esc.impl;
8  di_esc_l: DEVICE d_esc.impl;
9  di_servo_r: DEVICE d_servo.impl;
10 di_servo_l: DEVICE d_servo.impl;
11 di_gps: DEVICE d_gps.impl;
12 di_sonar: DEVICE d_sonar.impl;
13 di_imu: DEVICE d_imu.impl;
14 CONNECTIONS
15 C1: PORT di_gps.position -> pi_sen_act.position;
... Here goes all others connections
(lines 16 to 35)
35 END UAV.impl;

```

Fig. 39 contains the expansion of the sensing and actuation process (*pi\_sen\_act* line 5 in the Fig. 38). It is possible to observe the set

<sup>1</sup><http://provant.paginas.ufsc.br>

of threads that is responsible for interfacing with these devices, sending the control references to actuators (thread *ti\_signalTransformation* line 5) and providing the system behavior estimation (threads *ti\_sensing* and *ti\_positionEst* lines 3 and 4).

Figure 39 – AADL representation of sensing and actuation process.

```

1 PROCESS IMPLEMENTATION p_est_act.impl
2   SUBCOMPONENTS
3     ti_sensing: THREAD t_sensing.impl;
4     ti_positionEst: THREAD t_positionEst.impl;
5     ti_signalTransformation: THREAD t_signalTransformation.impl;
6   CONNECTIONS
7     C1: PORT distance -> ti_sensing.distance;
... Here goes all others connections (lines 8 to 25)
26 END p_est_act.impl;

```

The designed architectural model allows the design team to evaluate some system properties such as, schedulability, inversion of priorities, and information flow. Physical characteristics could also be validated in this model such as weight, power consumption, and others.

Regarding the proposed method, two different phases are described, defining the evaluation of task properties and system schedulability respectively. Details related to the application of these phases on the UAV design process are described next.

### 6.3.1 Evaluation of tasks behavior and interferences

Initially, the set of system tasks need to be split into a set of sub-representations that will be individually evaluated and validated. Each representation preserves the system and task-related process, as well as the set of devices related to the task. Figures 40 and 41 describe a subset of architectural representation, defining the required structures to estimate the UAV altitude based on the GPS information.

Based on this information, the set of possible failures associated with the system devices is defined as a fault-tree (Activity 1). With regard to the UAV devices, the fault-trees were defined for the servomotors, Electronic Speed Controllers (ESCs), Inertial Measurement Unit (IMU), Global Positioning System (GPS), and Sonar. Fig. 42 presents the designed fault-tree for the GPS sensor, containing the information extracted from its datasheet (NOVATEL, 2014).

Considering our focus on the design of UAV software

Figure 40 – Split of UAV model with a subset of devices.

```

1 SYSTEM IMPLEMENTATION UAV.impl
2   SUBCOMPONENTS
3     --PROCESS
4     pi_sen_act: PROCESS p_sen_act.impl;
5     --DEVICE
6     di_gps: DEVICE d_gps.impl;
7   CONNECTIONS
8     C1: PORT di_gps.position -> pi_sen_act.position;
9     C2: PORT pi_sen_act.requisition -> di_gps.requisition;
10 END UAV.impl;

```

Figure 41 – AADL representation of position estimation components.

```

1 PROCESS IMPLEMENTATION p_est_act.impl
2   SUBCOMPONENTS
3     ti_positionEst: THREAD t_positionEst.impl;
4   CONNECTIONS
5     C1: PORT position -> ti_positionEst.position;
6     ... Here goes all others connections (lines 6 to 8)
7   END p_est_act.impl;

```

components, at this time only the logical failures are considered (dark blue blocks of Fig. 42). These failures are flagged by the device, sending in the message package into each type of reported error. The designed faults-trees are also based on the works related to UAV faults described in (CASWELL; DODD, 2014; FUGGETTI; GHETTI; ZANZI, 2015).

Based on the set of defined sub-representations, the specified fault properties are integrated into each sub-model by the use of AADL EA, increasing the model detail (Activity 2). Fig. 43 shows the GPS failures according to the fault-tree.

Regarding the behavior specification coupled with the devices error properties, the threads behavior properties are analyzed and extended if required (Activity 3). In this activity the designers detail various properties such as threads states, transitions, guards, the subprograms access, among others. The proposed design conventions can be used at this time, in order to provide a proper mapping model.

The behavioral characteristics are presented in Fig. 44, detailing: the thread variables (line 6); the set of execution states (lines 8 to 10); and the set of system transitions (lines 12 to 27).

Detailing the base model the formal verification process can be started (Activity 4). In this way, based on the UAV AADL model the UPPAAL timed automata is generated (Activity 4.1). This process is performed on each generated sub-representation, representing details of



Figure 42 – GPS fault tree representation.

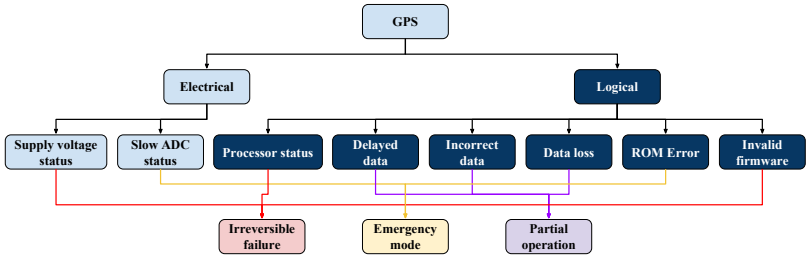


Figure 43 – AADL GPS error representation.

```

1 ERROR BEHAVIOR gpsError
2 EVENTS
3   processorError : error event; delayedData : error event;
4   incorrectData : error event; dataLoss : error event;
5   romError : error event; invalidFirmware : error event;
6 STATES
7   operational : initial state; partialOperation: state;
8   emergencyMode : state; irreversibleFailure : state;
9 TRANSITIONS
10  T1 : operational -[ delayedData ]-> partialOperation;
11  T2 : operational -[ incorrectData ]-> partialOperation;
12  T3 : operational -[ dataLoss ]-> partialOperation;
13  T4 : operational -[ romError ]-> emergencyMode;
14  T5 : operational -[ processorError ]-> irreversibleFailure;
15  T6 : operational -[ invalidFirmware ]-> irreversibleFailure;
16 END BEHAVIOR;

```

the UAV set of threads (actuation, sensing, and position estimation), and the set of system devices.

Applying the transformation process, the UAV system is mapped into a set of templates. Fig. 45 depicts a template structure representing the UAV's position thread that is responsible for providing the GPS interface and estimating the system position.

The set of predefined properties detailed in the AADL model can be observed in this template. Such characteristics describe the devices interface, among others. Regarding the UAV devices, the GPS sensor has an interface with the presented thread. In this sense, the generated template is shown in Fig. 46.

Regarding the GPS model the set of main execution states is observed (*Idle*, *Processing*), *PartialOperation*, *EmergencyMode*, and *IrreversibleFailure*, as well as being coupled with these states the predefined set of possible failures and their probabilities of occurrence

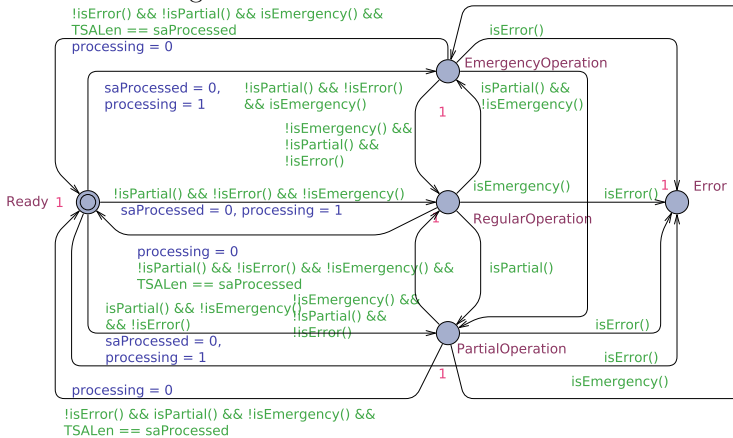
Figure 44 – AADL position thread behavior.

```

1 THREAD IMPLEMENTATION t_positionEst.impl
2 PROPERTIES
4 ANNEX BEHAVIOR_SPECIFICATION {**
5 VARIABLES
6   saProcessed : integer; processing : integer;
7 STATES
8   readyState : initial state; error : complete final state;
9   regularOperation : state; partialOperation : state;
10  emergencyOperation : state;
11 TRANSITIONS
12  T1 : readyState -[!isPartial() && !isError() && !isEmergency()]->
regularOperation {saProcessed := 0; processing := 1});
... here goes others thread transition lines 13 to 27.
28 END t_positionEst.impl

```

Figure 45 – Position estimation task.

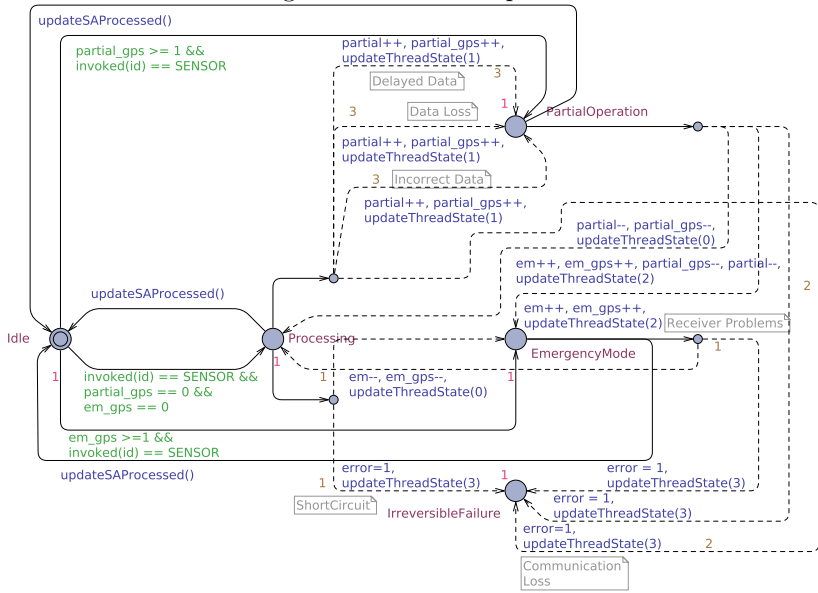


are presented.

Once the automata representations are generated, the designer needs to formally define the set of properties that will be evaluated (Activity 4.2), thus they need to define them as UPPAAL queries. These queries are written in TCTL and detail properties such as reachability, safety, and deadlock freeness for example. Examples of queries are presented in Section 6.4.

In terms of the system analysis, the UPPAAL tool performs a state space exploration to validate the designed queries. The system evaluation also includes queries related to defined probabilities, by the use of UPPAAL-SMC. Coupled with the task individual validation of the scheduling analysis is performed to validate the set of proposed

Figure 46 – GPS template.



tasks, these characteristics are detailed in the following section.

### 6.3.2 Schedulability Analysis

Based on the top-level representation from Fig. 38, the designers need to prepare the AADL model to be analyzed. Initially the defined tasks of this model need to be refined integrating some behavioral properties, such as period, deadline, execution time, priority, among others (Activity 1). The behavioral characteristics are presented in Fig. 44, detailing: the thread periodicity; the execution time; its period; the thread priority; its deadline (lines 3 and 5); the thread variables (line 8); the set of execution states (lines 10 to 11); and the set of system transitions (lines 13 to 31).

Aiming to support the scheduling analysis two additional structures need to be included, describing the system scheduler and the UAV behavior representation. These structures are automatically included during the transformation process, the designers just need to include two devices on the AADL model then name them scheduler and

Figure 47 – AADL position thread behavior.

```

1 THREAD IMPLEMENTATION t_positionEst.impl
2 PROPERTIES
3   dispatch_protocol => periodic;
4   compute_execution_time => 9000 us .. 10000 us;
5   period => 100000 us; Priority => 3; deadline => 100000 us;
6 ANNEX BEHAVIOR_SPECIFICATION {**
7   VARIABLES
8     t : clock; ax : clock;
9   STATES
10  readyState : initial state; error : complete final state; idle : state;
11  blocked : state; processing : state; partial : state; emergency : state;
12  TRANSITIONS
13  T1 : readyState -[]-> processing {ax := 0; run?;
guard!("head()", "=", "id");};
14  T2 : processing -[ax>=C[id]]-> idle {done!; remove();
guard!("head()", "=", "id");};
... here goes others thread transition lines 15 to 31.
32 END t_positionEst.impl

```

behavior.

Detailing the base model the formal verification process can be started (Activity 2). In this way, based on the UAV AADL model, the UPPAAL timed automata is generated (Activity 2.1). The generated representation details the UAV structure, the set of threads (actuation, sensing, and position estimation), and the set of system devices.

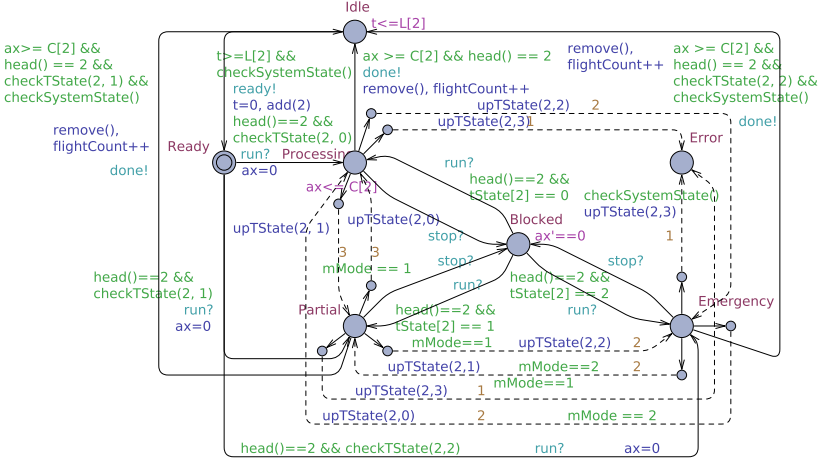
Applying the transformation process, the UAV system is mapped into a set of templates. Fig. 48 depicts a task template structure, that is used as a basis for the schedulability analysis.

The set of properties detailed in the AADL model can be observed in this template, including the thread's period, the devices interface, the scheduler functions, among others.

Once the automata representations are generated, the designer needs to formally define the set of properties that will be evaluated (Activity 2.2), thus they need to define them as UPPAAL queries. These queries are written in TCTL and can specify properties such as reachability, safety, and deadlock freeness for example. Examples of such queries are also presented in Section 6.4.

In terms of the system analysis, the UPPAAL tool performs a state space exploration to validate the designed queries (Activity 2.3). The system evaluation also includes queries related to defined probabilities, by the use of UPPAAL-SMC. The details related to the performed system analysis on the UAV system are detailed in the following section.

Figure 48 – Tasks template.



## 6.4 UAV PROPERTIES EVALUATION

In this section we present preliminary experiments that were carried out in the UAV system while formulating the design method described in this paper. These experiments involved mostly two efforts: i) the construction of the various timed automata which we presented in Fig. 45, Fig. 46, and Fig. 48, capturing respectively the GPS properties and the task characteristics on the two proposed scenarios; ii) the specification of the relevant properties of the UAV model covering safety, liveness, respect of deadlines, and causes for deadlocks.

In the rest of this section some examples of such UPPAAL specifications that were developed and checked are presented.

**Example-Spec 1** *If a task is running, then all the other tasks have to be either blocked, idle, ready, or in an error state, that is, only one task can be running at a time. This property was checked by specifying the following formula scheme:*

$$A[] \text{ (forall } (i : \text{int}[0, N]) \text{ forall } (j : \text{int}[0, N]) \text{ not } (\psi_i \text{ and } \psi_j \text{ imply } (i \neq j))), \quad (6.1)$$

where  $N$  is the set of system tasks, and  $\psi_i = T_i.\text{Running}$  and  $\psi_j = T_j.\text{Running}$  where  $T_i$  and  $T_j$  are tasks and  $\text{Running}$  denotes its executing states.

**Example-Spec 2** *All tasks run at least once, and therefore reach a state where they are idle. This liveness property is, as expected, only partially fulfilled since a task can reach an error state (e.g., due to a device or a function failure) during its execution. It is expressed by the following UPPAAL formula, where  $T_1 \dots T_k$  denote all the model tasks.*

$$E \langle \rangle (T_1.Idle \text{ and } \dots \text{ and } T_k.Idle), \quad (6.2)$$

**Example-Spec 3** *Whatever the task we consider, that task is executed only if the scheduler is either running or processing a new request. This safety condition ensures therefore that it does not exit tasks running out of the scheduler control, and is expressed as:*

$$A[] \text{ not } (\phi \text{ and not } (Scheduler.Run \text{ or } Scheduler.NewRequest)), \quad (6.3)$$

*such that  $\phi$  specifies all possible task states that correspond to its execution. The way to state this for a specific  $T_i$  is through the term*

$$\phi = (T_i.State_1 \text{ or } \dots \text{ or } T_i.State_k), \quad (6.4)$$

*where  $T_i$  represents a system thread,  $i$  represents a task in  $[0, N]$  and  $N$  define the number of tasks. The  $State_k$  denotes their execution states (which are a subset of task execution states).*

**Example-Spec 4** *Considering the set of system tasks, a task is running only if its execution time is smaller than its deadline. This is expressed by the following formula:*

$$A[] \text{ not } (\text{forall } (i : \text{int}[0, N])((T_i.State_1 \text{ or } \dots \text{ or } T_i.State_k) \text{ and } (T_i.ax > D[i]))) \quad (6.5)$$

*where  $T_i$  represents system threads,  $D[i]$  is the prescribed deadline, the  $ax$  field is the current total execution time, and each  $State_1, \dots, State_k$  represents task execution states.*

**Example-Spec 5** *A system deadlock is possible only if one of the system threads is on error state. This property was checked by specifying the formula scheme:*

$$A[] \text{ deadlock imply } (T_1.Error \text{ or } \dots \text{ or } T_i.Error) \quad (6.6)$$

*with  $T_1 \dots T_i$  denoting the tasks of the system, and **deadlock** is UPPAAL's keyword that denotes that there is a deadlock in the model.*

**Example-Spec 6** *As a final example we define a specification that brings statistical analysis of the model. For this, we used the Statistical Model Checking (SMC) facilities whose version of UPPAAL we have adopted.*

*This specification refers to the probability of the system to reach an actuator error state due to its execution. This probability condition is expressed by the formula:*

$$\Pr [\leq 12000] (\langle\langle \text{Actuator}(i).\text{EmergencyMode} \rangle\rangle), \quad (6.7)$$

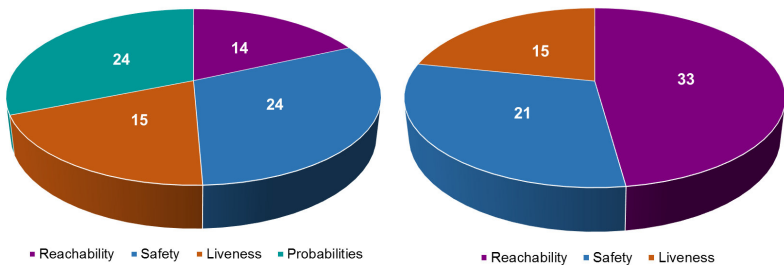
*where the bound defined ( $\leq 12000$ ) represents the thread period that interfaces with this actuator and  $\text{Actuator}(i)$  denotes the  $i^{\text{th}}$  system actuator. For instance, when considering  $\text{Actuator}(0)$  (named ESC right in the UAV model), this property is satisfied with a probability in  $[0.107051, 0.206887]$  with 0.95 certainty.*

Regarding the UAV properties evaluation, overall, one hundred and forty-six (146) properties were analyzed in different categories including, reachability, safety, liveness, and deadlock freeness. This set of properties are split in the two proposed phases (Tasks evaluation and schedulability analysis), being seventy seven applied to evaluate the first phase and sixty nine defined to evaluate the second analysis phase. The properties are arranged into the defined categories as depicted in Fig. 49. As it can be observed, in the second phase of analysis the use of probabilities is not applied.

Figure 49 – Evaluated UAV properties.

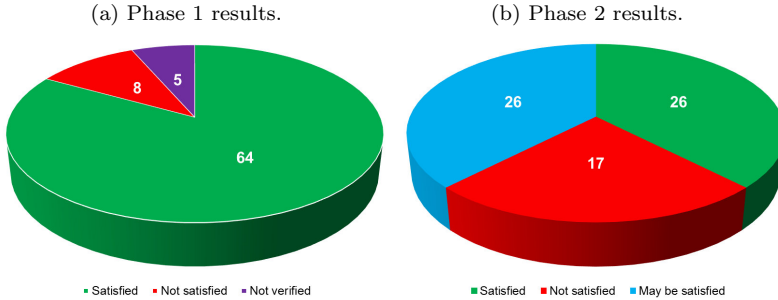
(a) Phase 1 - defined properties.

(b) Phase 2 - defined properties.



Defined the set properties the UAV system its evaluated by performing formal verification. The obtained results of these analysis is presented in Fig. 50.

Figure 50 – Obtained results of evaluated UAV properties.



Analyzing the Phase 1 obtained results it is observed that 83,12% of the evaluated properties are satisfied, 10,39% are not satisfied, and 6,49% are not verified. Another aspect regards the probabilities evaluation where twenty four properties are defined providing estimations with 95% certainty.

Observing Phase 2 results it is verified that 37,68% of the evaluated properties are satisfied, 37,68% may be satisfied, and 24,64% are not satisfied. These results are according to our expectations due to the specifications nature, which may not be satisfied is related to fact that the models also represent the error occurrence on the system execution.

The obtained results with the UAV model showed that the system meets its requirements. This implies that the system threads meet their deadlines, the safety properties are satisfied, and no deadlocks are found except in the event of an error occurring.

## 6.5 SUMMARY

In this chapter a method devoted to integrate formal verification in the UAV design process (proposed in Chapter 4) is presented. It consists of performing a model transformation of AADL models to a timed automata representation that is suitable for analysis using MC. The transformation process is supported by a developed tool named *ECPS Verifier* and MC can be performed using the UPPAAL tool. The tool supports the application of the design activities promoted by the authors, improving the architectural model characteristics, and



providing a means by which to extract the components behavior, that is evaluate by performing the MC formal method.

In comparison to the related works, the partial compliance of their transformation process in respect to the designed architectural models is observed. Most of the time these models require manual intervention from the designers in order to integrate additional features such as thread behavior and error properties. In this sense, the present proposal provides a means by which to represent the system characteristics to support the complete timed automata extraction and its evaluation.



## 7 CONCLUSIONS

In this PhD thesis contributions were presented to improve the CPS design processes, especially applied to design UAV applications. The proposed contributions are designed after extensive analysis of different CPS design methods, taking into account the required UAV characteristics and activities suitability for its design. Studies were also performed to evaluate the characteristics related to the integration of the system devices on the design process of these applications. The integration of formal verification methods applied to these process was also evaluated. Based on the performed studies it was observed that there is not a definitive solution that can cover the UAV design specifics, integrating the system devices, and validating its behavior.

In this context three main contributions were proposed including: i) the proposal of a design method, applied to systematize the UAV design activities; ii) an extension of a transformation process from functional model to architectural model, integrating an intermediary step that provides the representation of system devices characteristics; iii) a transformation process from architectural model to timed automata representations, to allow the system formal evaluation.

Initially a design method devoted to UAV built was proposed, this process aims to fill some gaps that were observed in others existing design methods. This method highlights the particularities related to UAV design, such as the mission design. Based on the proposed method, activities are also suggested to represent some system characteristics which are not so widely discussed by other authors, such as the specifications of the system sensors and actuator properties, and the integration of formal method directly on the design method.

Once the propose method was finished, it was possible to observe that this method may be extended and generalized to cover the design of different CPS applications. However, additional work is need therefore. For such reason, and also to limit the scope of this thesis, any adaptation or generalization of the current proposal is left for future works. Another observed point regards the implementation phase, which was not so widely discussed. In this sense, some activities mainly related to the UAV construction were not properly detailed, as well as characteristics related to the costumers product validation are not presented. In addition it is observed that the process to generate the application code is considered implicit in some project phases, and due to this fact it is not widely discussed on this method.

In this way, aiming to specialize the proposed design process, and providing the representation and integration of the set of device characteristics on the project, a transformation process was proposed. This process is based on the functional representation (Simulink model) and provides an architectural model (AADL model). The proposed transformation process extends the work from (PASSARINI, 2014). However, as discussed in Chapter 5 Passarini's proposal did not cover some design steps like the integration of system devices. In this way, an intermediary step was created to ensure the specification of devices information before performing the transformation process. The *ECPS Modeling* tool was designed to support this transformation process, providing the generation of architectural models including the device's characteristics.

The second improvement applied to the proposed design method is with regard to the integration of formal verification techniques on the UAV design process. The selected technique was model checking, based on its expressiveness and its utilization in different environments, providing a means to validate the applications properties. To support this technique application the UPPAAL tool was chosen, and a tool named *ECPS Verifier* was design to support the transformation process from AADL language to UPPAAL timed automata.

In nutshell this thesis provide an integrated solution representing the UAV design process phases and activities, that is complemented by two transformation processes that aim to enable the system device characteristics integration, and the formal evaluation of the system properties. The proposed tools are designed as plug-ins to the OSATE tool, a framework that supports the design of AADL models. The plug-ins can be obtained respectively on <https://github.com/fernandosgoncalves/ECPSModeling>, and <https://github.com/fernandosgoncalves/ECPSVerifier>.

The proposed method and tools were applied to the design of a tilt-rotor UAV. The results showed that by applying the proposed steps complementary representations are generated, ensuring more information in the project. By performing the transformation processes we intend to reduce the project design time, as well as the process being considered less prone to errors.

## 7.1 FUTURE WORKS

In the following a list of possible future works related to this dissertation, is proposed:

- Review the proposed UAV design method, improving the method characteristics to support the design of different CPS applications. A V-model should be applied to the proposed method, increasing the relationship between the system design, the verification, and tests.
- Extend the *ECPS Modeling* tool to integrate more information during the transformation process, providing functionalities such storage of device characteristics and importation of this information from a specific file.
- Improve the designed transformation engine from *ECPS Verifier* tool, in order to increase the AADL language components coverage. In the same way, evaluate the use of different sources to the transformation engine for example an XML file generated after instantiate the system for example.
- Working in the direction of integrating dynamic verification methods to complement the model checking activities, as there might be cases of properties that cannot be verified statically, so it may be possible to enforce upon run-time. Thereby it is necessary to enrich the target model with new components that assume the form of runtime monitors (generated from some formal specifications). The resulting enriched model will still be subject to the same automata generation principles of the presented work, making sure that the model together with the monitors still satisfies the original specifications. In this sense, system properties can be evaluated during the applications execution, by the use of techniques like Runtime Monitoring.



## BIBLIOGRAPHY

AHMED, S. H.; KIM, G.; KIM, D. Cyber Physical System: Architecture, applications and research challenges. **2013 IFIP Wireless Days (WD)**, p. 1–5, 2013.

ALANEN, M. et al. Model driven engineering: A position paper. In: FERNANDES, J. a. M. et al. (Ed.). **Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'04)**. [S.l.]: Turku Centre for Computer Science, 2004. p. 25–29.

ALMENARA, A. **SAMU Maringá Recebe Nova Viatura Para Suporte Médico Avançado**. 2017. Available from Internet: <<http://www.andrealmenara.com.br/noticia/ler/samu-de-maringa-recebe-nova-viatura-para-suporte-medico-avancado>>.

ALUR, R. **Principles of Cyber-Physical Systems**. [S.l.]: MIT Press, 2015. ISBN 9780262029117.

ALUR, R.; COURCOUBETIS, C.; DILL, D. Model-checking for real-time systems. In: [1990] **Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science**. [S.l.: s.n.], 1990. p. 414–425.

BACK, N. et al. **Projeto integrado de produtos: Planejamento, Concepção e Modelagem**. Barueri: Manole, 2008. 601 p. ISBN 9788520422083.

BAIER, C.; KATOEN, J.-P. **Principles Of Model Checking**. [s.n.], 2008. I–XVII, 1–975 p. ISSN 00155713. ISBN 9780262026499. Available from Internet: <<http://mitpress.mit.edu/books/principles-model-checking>>.

BAO, Y. et al. Quantitative Performance Evaluation of Uncertainty-Aware Hybrid AADL Designs Using Statistical Model Checking. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, vol. 0070, n. c, p. 1–1, 2017. ISSN 0278-0070. Available from Internet: <<http://ieeexplore.ieee.org/document/7875425/>>.

BAUDRY, B. et al. Model transformation testing challenges. In: **Proceedings of the IMDDMDT workshop at ECMDA'06**.

Bilbao, Spain: [s.n.], 2006. Available from Internet:  
 <<http://www.irisa.fr/triskell/publis/2006/baudry06b.pdf>>.

BECKER, L. B. et al. Development process for critical embedded systems. In: **WSE 2010, Gramado. Anais. Porto Alegre: SBC.** [S.l.: s.n.], 2010. p. 95–108.

BEHRMANN, G.; DAVID, A.; LARSEN, K. G. A Tutorial on Uppaal. In: **Lecture Notes in Computer Science.** Springer Berlin Heidelberg, 2004. p. 200–236. ISBN 978-3-540-23068-7. Available from Internet: <[http://link.springer.com/10.1007/978-3-540-30080-9\\_7](http://link.springer.com/10.1007/978-3-540-30080-9_7)>.

BEHRMANN, G.; DAVID, A.; LARSEN, K. G. Formal methods for the design of real-time systems: International school on formal methods for the design of computer, communication, and software systems. In: \_\_\_\_\_. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. cap. A Tutorial on Uppaal, p. 200–236. ISBN 978-3-540-30080-9.

BENGTSSON, J. et al. Uppaal — a Tool Suite for Automatic Verification of Real-Time Systems. **Workshop on Verification and Control of Hybrid Systems III**, n. 1066, p. 232–243, 1995.

BERKELEY, U. **Ptolemy Project.** 1999.  
<http://ptolemy.eecs.berkeley.edu/>.

BERTHOMIEU, B. et al. Real-Time Model Checking Support for AADL. mar 2015. Available from Internet:  
 <<http://arxiv.org/abs/1503.00493>>.

BERTHOMIEU, B.; RIBET, P. O.; VERNADAT, F. The tool tina ? construction of abstract state spaces for petri nets and time petri nets. **International Journal of Production Research**, vol. 42, n. 14, p. 2741–2756, 2004. Available from Internet:  
 <<http://dx.doi.org/10.1080/00207540412331312688>>.

BOHEM, B. W. Software Engineering; R & D trends and defense needs. In: WEGNER, P. (Ed.). **Research Directions in Software Technology (Ch. 22).** Cambridge, MA: MIT Press, 1979. p. 1–9.

BOZGA, M. et al. Kronos : A Model-Checking Tool for Real-Time Systems \*. **Springer International Journal of Software Tools for Technology Transfer**, vol. 1, 1997.

BOZZANO, M. et al. The COMPASS approach: Correctness, modelling and performability of aerospace systems. **LNCS**, p. 173–186, 2009. ISSN 03029743.



BRADE, T. et al. Model-Driven Development of Embedded Systems. **12th Brazilian Workshop on Real-Time and Embedded Systems**, p. 12, 2010. Available from Internet:

<<http://www.wseas.us/e-library/conferences/2010/Cambridge/SEPADS/SEPADS-17.pdf>>.

CAI, G. et al. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. In: **Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on**. [S.l.: s.n.], 2008. p. 29–34.

CAI, G. et al. Systematic design methodology and construction of {UAV} helicopters. **Mechatronics**, vol. 18, n. 10, p. 545 – 558, 2008b. ISSN 0957-4158.

CASWELL, G.; DODD, E. Improving UAV Reliability. n. 301, p. 7, 2014.

CHANDHOKE, S. et al. A model-based methodology of programming cyber-physical systems. In: **2011 7th International Wireless Communications and Mobile Computing Conference**. [S.l.: s.n.], 2011. p. 1654–1659. ISSN 2376-6492.

CHEN, W. L. et al. Researches on robot system architecture in CPS. **2015 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, IEEE-CYBER 2015**, n. 2013921069, p. 603–607, 2015.

CLARKE, E.; GRUMBERG, O.; PELED, D. **Model Checking**. MIT Press, 1999. ISBN 9780262032704. Available from Internet: <<https://books.google.com.br/books?id=Nmc4wEaLXFEC>>.

CONQUET, E. et al. The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software. In: **European Congress on Embedded Real-Time Software (ERTS 2010)**. Toulouse, France: [s.n.], 2010. p. 1–10.

CORREA, T. et al. Supporting the Design of Safety Critical Systems Using AADL. **Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS**, p. 53–62, 2010.

COSTA, F. et al. The use of unmanned aerial vehicles and wireless sensor network in agricultural applications. In: **Geoscience and**

**Remote Sensing Symposium (IGARSS), 2012 IEEE International.** [S.l.: s.n.], 2012. p. 5045–5048. ISSN 2153-6996.

DAVID, A. et al. UPPAAL SMC tutorial. **International Journal on Software Tools for Technology Transfer**, vol. 17, n. 4, p. 397–415, 2015. ISSN 14332787.

DELANGE, J. et al. An MDE-based Process for the Design, Implementation and Validation of Safety-Critical Systems. In: **Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS.** [S.l.: s.n.], 2010. p. 215–223. ISBN 9780769540153.

DERLER, P.; LEE, E. A.; VINCENTELLI, A. S. Modeling cyber-physical systems. **Proceedings of the IEEE**, n. 1, p. 13–28, Jan 2012. ISSN 0018-9219.

DI NATALE, M. et al. An MDA Approach for the Generation of Communication Adapters Integrating SW and FW Components from Simulink. **Model-Driven Engineering Languages and Systems**, vol. 8767, p. 353–369, 2014. ISSN 16113349. Available from Internet: <[http://link.springer.com/chapter/10.1007/978-3-319-11653-2\\_22%5Cnhttp://link.springer.com/10.1007/978-3-319-11653-2](http://link.springer.com/chapter/10.1007/978-3-319-11653-2_22%5Cnhttp://link.springer.com/10.1007/978-3-319-11653-2)>.

DOERING, D. A Model Driven Engineering Methodology for Embedded System Designs-HIPAO2. **2014 12Th Ieee International Conference on Industrial Informatics (Indin)**, p. 787–790, 2014. ISSN 1935-4576.

ECLIPSE. **Eclipse.** 2000. <https://eclipse.org/>.

ECLIPSE. **Eclipse Modeling Framework.** 2004. <http://www.eclipse.org/emf>.

ECLIPSE. **Linguagem ATL.** 2006. <https://wiki.eclipse.org/ATL/Concepts>.

FARAIL, P.; GAUFILLET, P. Topcased: un environnement de développement open source pour les systèmes embarqués. **Génie logiciel**, GL & IS, Meudon, France, p. 16–20, 5 2005.

FARAIL, P. et al. The topcased project: a toolkit in open source for critical aeronautic systems design. **European Congress on Embedded Real-Time Software (ERTS 2006)**, p. 8, 1 2006.

FEILER, P. H.; GLUCH, D. P. **Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language**. 1st. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321888944, 9780321888945.

FEILER, P. H.; GLUCH, D. P.; HUDAK, J. J. **The Architecture Analysis & Design Language (AADL): An Introduction**. [S.l.], 2006.

FUGGETTI, G.; GHETTI, A.; ZANZI, M. Based on Handy FDI Current Sensor and a Fail- Safe Configuration of Control Surface Actuators. p. 356–361, 2015.

GAAEVIC, D. et al. **Model Driven Architecture and Ontology Development**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 3540321802.

GARAVEL, F. L. H.; MATEESCU, R. **An overview of CADP 2001**. ago. 2001. 13–24 p.

GONÇALVES, F. S. et al. Vant autônomo capaz de comunicar com uma rede de sensores sem fio. In: **X Congresso Brasileiro de Agroinformática (SBIAGRO 2015)**. Ponta Grossa - PR: [s.n.], 2015.

GONÇALVES, F. S.; BECKER, L. B. Preparing cyber-physical systems functional models for implementation. In: **V Brazilian Symposium on Computing System Engineering (SBESC 2015)**. Foz do Iguaçu - PR: [s.n.], 2015.

GONÇALVES, F. S.; BECKER, L. B. Model driven engineering approach to design sensing and actuation subsystems. In: **2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.: s.n.], 2016. p. 1–8.

GONÇALVES, F. S.; BECKER, L. B.; RAFFO, G. V. Managing CPS complexity: Design method for Unmanned Aerial Vehicles. In: **2016 1st IFAC Conference on Cyber-Physical & Human-Systems**. [S.l.: s.n.], 2016.

GONÇALVES, F. S. et al. Formal verification of aadl models using uppaal. In: **VII Brazilian Symposium on Computing Systems Engineering (SBESC 2017)**. Curitiba - PR: [s.n.], 2017.

- GONCALVES, F. S. et al. Assessing the use of simulink on the development process of an unmanned aerial vehicle. In: **3rd Workshop on Cyber-Physical Systems (CyPhy 2013)**. [S.l.: s.n.], 2013b.
- HALBWACHS, N. A synchronous language at work: the story of lustre. In: **Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on**. [S.l.: s.n.], 2005. p. 3–11.
- HALBWACHS, N. et al. The synchronous data flow programming language lustre. **Proceedings of the IEEE**, vol. 79, n. 9, p. 1305–1320, Sep 1991. ISSN 0018-9219.
- HALBWACHS, N.; RAYMOND, P. **A Tutorial Of Lustre**. 2001.
- HAMDANE, M. E.; CHAOUI, A.; STRECKER, M. From AADL to timed automaton-A verification approach. **International Journal of Software Engineering and its Applications**, vol. 7, n. 4, p. 115–126, 2013. ISSN 17389984.
- HARRISON, W. L. et al. Model-driven engineering from modular monadic semantics: Implementation techniques targeting hardware and software. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, vol. 5658 LNCS, p. 20–44, 2009. ISSN 03029743.
- HARTMANN, P. A. et al. A framework for generic HW/SW communication using remote method invocation. In: **2011 Electronic System Level Synthesis Conference (ESLsyn)**. [S.l.]: IEEE, 2011. p. 1–6. ISBN 978-1-4577-0634-9.
- HENDERSON-SELLERS, B. **On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages**. Springer, 2012. (SpringerBriefs in Computer Science,). Available from Internet: <<http://dx.doi.org/10.1007/978-3-642-29825-7>>.
- HENZINGER, T. A.; HO, P.-h.; WONG-TOI, H. HyTech: The Next Generation\*. In: **Real-Time Systems Symp.** [S.l.: s.n.], 1995.
- HESSE, W. More matters on (meta-)modelling: remarks on Thomas Kühne's "matters". **Software & Systems Modeling**, Springer, vol. 5, n. 4, p. 387–394, 2006. ISSN 1619-1366. Available from Internet: <<http://dx.doi.org/10.1007/s10270-006-0033-9>>.

HU, K. et al. Exploring AADL verification tool through model transformation. **Journal of Systems Architecture**, vol. 61, n. 3-4, p. 141–156, 2015. ISSN 13837621.

HUTH, M.; RYAN, M. **Logic in Computer Science**. [s.n.], 2004. 440 p. ISSN 1098-6596. ISBN 9783540425540. Available from Internet: <[http://bilder.buecher.de/zusatz/12/12609/12609080 lese\\_1.pdf](http://bilder.buecher.de/zusatz/12/12609/12609080 lese_1.pdf)>.

INRIA. **AADL to Signal/SSME Transformation**. [S.l.], 05 2012.

ISMAIL, H. I. et al. DSVerifier: A bounded model checking tool for digital systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, vol. 9232, p. 126–131, 2015. ISSN 16113349.

JENSEN, J. C.; CHANG, D. H.; LEE, E. A. A model-based design methodology for cyber-physical systems. In: **Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International**. [S.l.: s.n.], 2011. p. 1666–1671.

JENSEN, J. C.; CHANG, D. H.; LEE, E. A. A model-based design methodology for cyber-physical systems. In: **IWCMC 2011**. [S.l.: s.n.], 2011. p. 1666–1671.

KATOEN, J.-P. et al. The ins and outs of the probabilistic model checker {MRMC}. **Performance Evaluation**, vol. 68, n. 2, p. 90–104, 2011. ISSN 0166-5316. Available from Internet: <<http://www.sciencedirect.com/science/article/pii/S0166531610000660>>.

KEANE, J. F.; CARR, S. S. A brief history of early unmanned aircraft. **The Johns Hopkins APL Technical Digest**, vol. 32, n. 3, p. 558–571, 2013.

KENT, S. Model driven engineering. In: **Proceedings of the Third International Conference on Integrated Formal Methods**. London, UK, UK: Springer-Verlag, 2002. (IFM '02), p. 286–298.

KLEPPE, A. G.; WARMER, J.; BAST, W. **MDA Explained: The Model Driven Architecture: Practice and Promise**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 032119442X.

KÜHNE, T. Clarifying matters of (meta-) modeling: an authors reply. **Software and Systems Modeling (SoSyM)**, Springer, vol. 5, n. 4, p. 395–401, December 2006. ISSN 1619-1366. Available from Internet: <<http://dx.doi.org/10.1007/s10270-006-0034-8>>.

LEE, E. Cyber physical systems: Design challenges. In: **Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on**. [S.l.: s.n.], 2008. p. 363–369.

LEE, E. A.; SESHIA, S. A. **Introduction to Embedded Systems - A Cyber-Physical Systems Approach**. 2. ed. [S.l.]: Lee and Seshia, 2015. <http://leeseshia.org/>. ISBN 978-1-312-42740-2.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. **J. ACM**, ACM, New York, NY, USA, vol. 20, n. 1, p. 46–61, jan. 1973. ISSN 0004-5411. Available from Internet: <<http://doi.acm.org/10.1145/321738.321743>>.

LOPES, D. et al. Interoperability of enterprise software and applications. In: \_\_\_\_\_. London: Springer London, 2006. cap. Mapping Specification in MDA: From Theory to Practice, p. 253–264. ISBN 978-1-84628-152-5. Available from Internet: <<http://dx.doi.org/10.1007/1-84628-152-0.23>>.

LOPES, D. et al. Metamodel matching: Experiments and comparison. In: **Software Engineering Advances, International Conference on**. [S.l.: s.n.], 2006. p. 2–2.

MAIA, N. E. N. **Odyssey-MDA: Uma Abordagem Para a Transformação de Modelos**. 105 p. Thesis (Ph.D.) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, Rj - Brasil, 3 2006.

MALCOLM, G. Behavioural equivalence, bisimulation, and minimal realisation. In: **Selected Papers from the 11th Workshop on Specification of Abstract Data Types Joint with the 8th COMPASS Workshop on Recent Trends in Data Type Specification**. London, UK, UK: Springer-Verlag, 1996. p. 359–378. ISBN 3-540-61629-2. Available from Internet: <<http://dl.acm.org/citation.cfm?id=645978.675845>>.

MARWEDEL, P. **Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems**. Springer

Netherlands, 2010. (Embedded Systems). ISBN 9789400702578.

Available from Internet:

<<https://books.google.com.br/books?id=EXboa4sXIRsC>>.

MASIN, M. et al. Cross-layer design of reconfigurable cyber-physical systems. In: **Design, Automation & Test in Europe**

**Conference & Exhibition (DATE), 2017**. Lausanne: IEEE, 2017. p. 740–745. ISBN 978-3-9815370-8-6. Available from Internet:

<<http://ieeexplore.ieee.org/document/7927088/>>.

MATHWORKS. **MATLAB**. 1994. <http://www.mathworks.com>.

MATHWORKS. **Mathworks MATLAB Simulink**. 2018.

<http://www.mathworks.com/products/simulink/>.

MELLOR, S.; CLARK, A.; FUTAGAMI, T. Model-driven development - guest editor's introduction. **Software, IEEE**, vol. 20, n. 5, p. 14–18, Sept 2003. ISSN 0740-7459.

MELLOR, S. J. et al. **MDA Distilled**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 0201788918.

MIRABILIS. **VisualSim Architect**. 2018.

<http://mirabilisdesign.com/new/visualsim/>.

MOON, I. Modeling programmable logic controllers for logic verification. **Control Systems, IEEE**, vol. 14, n. 2, p. 53–59, April 1994. ISSN 1066-033X.

MORELLI, M.; DI NATALE, M. An MDE approach for the design of platform-aware controls in performance-sensitive applications. **19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014**, 2014. ISSN 1946-0740.

NAIDOO, Y.; STOPFORTH, R.; BRIGHT, G. Development of an uav for search amp; rescue applications. In: **AFRICON, 2011**. [S.l.: s.n.], 2011. p. 1–6. ISSN 2153-0025.

NAVET, N. et al. **Lean Model-Driven Development through**

**Model-Interpretation: the CPAL design flow**. [S.l.], Out 2015.

10 p. Available from Internet: <<http://hdl.handle.net/10993/22279>>.

NOVATEL. **OEMStar Receiver - Firmware Reference Manual**.

6. ed. [S.l.], 2 2014. Available from Internet:

<<https://www.novatel.com/assets/Documents/Manuals/om-20000127.pdf>>.

OMG. **MDA Guide Version 1.0.1**. jun. 2003.

<http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>. Available from Internet:

<<http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>>.

ONEM, E.; GURDAG, A. B.; CAGLAYAN, M. U. Secure routing in ad hoc networks and model checking. In: TRAFFORD PUBLISHING. **Security of Information and Networks: Proceedings of the First International Conference on Security of Information and Networks (SIN 2007)**. [S.l.], 2008. p. 346.

PAMPAGNIN, P. et al. Model driven hardware design: One step forward to cope with the aerospace industry needs. **2008 Forum on Specification, Verification and Design Languages, FDL'08**, p. 179–184, 2008. Available from Internet:

<<http://www.scopus.com/inward/record.url?eid=2-s2.0-67650458333&partnerID=40&md5=94dd06e3ded8e3ba28ec6fb3337e7421>>.

PAPACHRISTOS, C. et al. Model predictive attitude control of an unmanned tilt-rotor aircraft. **Industrial Electronics (ISIE), 2011 IEEE International Symposium on**, p. 922–927, June 2011.

PASSARINI, R. et al. Cyber-physical systems design: transition from functional to architectural models. **Design Automation for Embedded Systems**, Springer US, p. 1–22, 2015. ISSN 0929-5585. Available from Internet:

<<http://dx.doi.org/10.1007/s10617-015-9164-y>>.

PASSARINI, R. F. **Transformação assistida de modelos**. Thesis (Ph.D.) — Universidade Federal de Santa Catarina, Florianópolis, SC - Brasil, 8 2014.

PING, J. et al. Generic unmanned aerial vehicle (uav) for civilian application-a feasibility assessment and market survey on civilian application for aerial imaging. In: **Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2012 IEEE Conference on**. [S.l.: s.n.], 2012. p. 289–294. ISSN 1985-5753.

PTOLEMAEUS, C. (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available from Internet: <<http://ptolemy.org/books/Systems>>.



RAHM, E.; BERNSTEIN, P. A. A survey of approaches to automatic schema matching. **The VLDB Journal**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, vol. 10, n. 4, p. 334–350, dez. 2001. ISSN 1066-8888. Available from Internet: <<http://dx.doi.org/10.1007/s007780100057>>.

RENAULT, X.; KORDON, F.; HUGUES, J. From AADL architectural models to petri nets : Checking model viability. **Proceedings of the 2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2009**, p. 313–320, 2009. ISSN 1555-0885.

ROUSSEL, J.-M.; LESAGE, J.-J. Validation and verification of grafceets using state machine. In: **IMACS-IEEE "CESA '96"**. Lille, France: [s.n.], 1996. p. pp. 758–764. Available from Internet: <<https://hal.archives-ouvertes.fr/hal-00353188>>.

SAE. SAE, **SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex**. 2011. <http://standards.sae.org/as5506/1>.

SAE. SAE, **SAE Architecture Analysis and Design Language (AADL) Annex Volume 2**. 2011. <http://standards.sae.org/as5506/2>.

SAE. **International Society of Automotive Engineers**. 2015. <http://www.sae.org/>.

SCHMIDT, D. C. Guest editor's introduction: Model- driven engineering. **Computer**, vol. 39, n. 2, p. 25–31, Feb 2006.

SEI. **Osate 2**. 2005. [https://wiki.sei.cmu.edu/aadl/index.php/Osate\\_2](https://wiki.sei.cmu.edu/aadl/index.php/Osate_2).

SEIDEWITZ, E. What models mean. **IEEE Software**, vol. 20, n. 5, p. 26–32, 2003.

SELIC, B. The pragmatics of model-driven development. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, vol. 20, n. 5, p. 19–25, set. 2003. ISSN 0740-7459. Available from Internet: <<http://dx.doi.org/10.1109/MS.2003.1231146>>.

SENSEFLY. **EBee the professional mapping drone**. 2018.  
<https://www.sensefly.com/drone/ebee-mapping-drone/>.

STEINBERG, D.; BUDINSKY, F.; MERKS, E. **EMF: Eclipse Modeling Framework**. Addison-Wesley, 2009. (Eclipse (Addison-Wesley)). ISBN 9780321331885. Available from Internet: <https://books.google.com.br/books?id=oAYcAAAACAAJ>>.

SUN, Z.; ZHOU, X. Extending and recompiling AADL for CPS modeling. **Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013**, p. 1225–1230, 2013.

UAVGLOBAL. **Bell Eagle Eye**. 2008. Available from Internet: <http://www.uavglobal.com/bell-eagle-eye/>>.

WANG, G. et al. Studying on aadl-based architecture abstraction of embedded software. In: **Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on**. [S.l.: s.n.], 2009. p. 14–19.

WANG, Y. et al. An AADL-Based Modeling Method for ARINC653-Based Avionics Software. In: **Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual**. [S.l.: s.n.], 2011. p. 224–229. ISSN 0730-3157.

XU, S. et al. Quantitative Analysis of Variation-Aware Internet of Things Designs Using Statistical Model Checking. In: **2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)**. IEEE, 2016. p. 274–285. ISBN 978-1-5090-4127-5. Available from Internet: <http://ieeexplore.ieee.org/document/7589807/>>.

YAN, Z. et al. From AADL to Timed Abstract State Machine: A Certified Model Transformation. **Journal of Systems and Software**, p. 42–68, 2014. ISSN 01641212.

YU, H. et al. Polychronous modeling, analysis, verification and simulation for timed software architectures. **Journal of Systems Architecture - Embedded Systems Design**, vol. 59, n. 10-D, p. 1157–1170, 2013.

YU, Z. et al. Research on Modeling and Analysis of CPS. In: CALERO, J. M. A. et al. (Ed.). **Autonomic and Trusted Computing: 8th International Conference, ATC 2011, Banff, Canada, September 2-4, 2011. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 92–105. ISBN 978-3-642-23496-5. Available from Internet: <[https://doi.org/10.1007/978-3-642-23496-5\\_7](https://doi.org/10.1007/978-3-642-23496-5_7)>.

ZHAO, Y.; MA, D. Embedded real-time system modeling and analysis using aadl. In: **Networking and Information Technology (ICNIT), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 247–251.