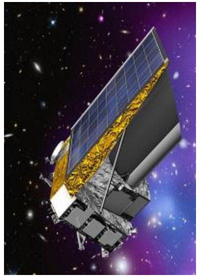
Model-Checking for TASTE designed Space Software Systems: Results and Lessons Learned

Iulia Dragomir
GMV

Model Based Space Systems and Software Engineering (MBSE) 2022

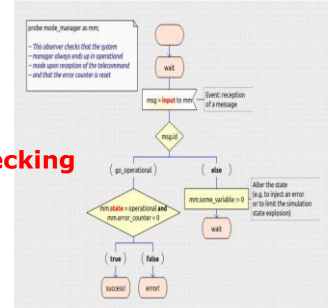
November 23rd, 2022

MoC4Space


(Courtesy of ESA)

⊨ ✓

IF Toolset model-checking



Introduction

- **Model-based Software/Systems Engineering** is an established development approach that enables:
  - Designing large, complex and heterogeneous systems with minimal effort and costs
  - Obtaining correct-by-construction implementation wrt system requirements with the help of (formal) V&V
- **TASTE** is an MBSE toolset from ESA that allows:
  - Real-time embedded software design
  - Software validation through static type analysis, real-time scheduling, interactive simulation and testing
  - Open topic: **formal V&V of TASTE** designs
    - ➢ Why formal V&V? E.g., Ariane 5, Hitomi, Boeing Starliner, …
    - ➢ ESA **MoC4Space project** (2021-2022) addressed this shortcoming by integrating a formal V&V approach based on model-checking in TASTE

Ariane 5 explosion, © ESA

TASTE

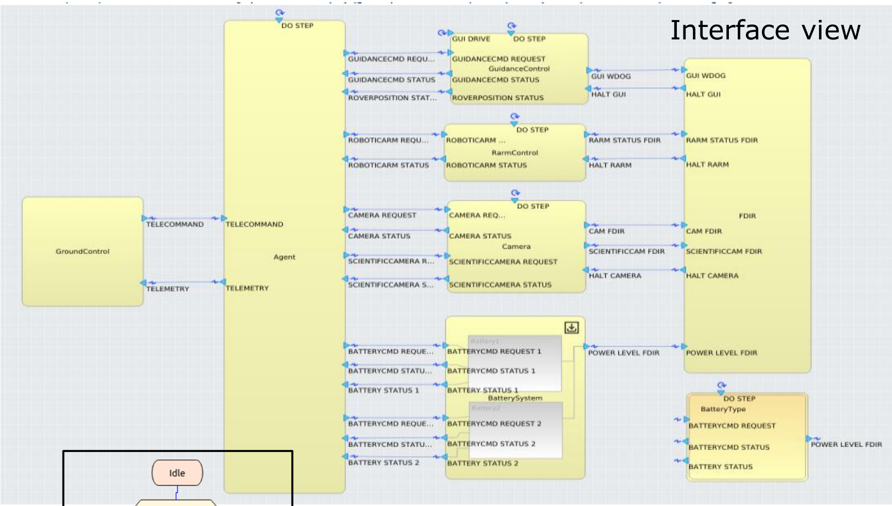
Excerpt from ERGO case study TASTE design

- Model-based development of heterogeneous, reactive, discrete embedded systems
- Uses several modelling formalisms (ASN.1, AADL , SDL, etc.) or programming languages (e.g., C)
- A **TASTE design** consists of:
  - **Data view** (in ASN.1)
  - **Hierarchical interface views** (software architecture and behaviour)
    - Communication is based on the notion of interfaces:
      - Cyclic: execute a behaviour at a certain frequency
      - Sporadic: whenever a request is received handle it
      - Protected: handle the request and provide an answer
    - Behaviour is either modelled as SDL state machines or implemented in C
  - **Deployment view**
  - **Concurrency view** computed from the above

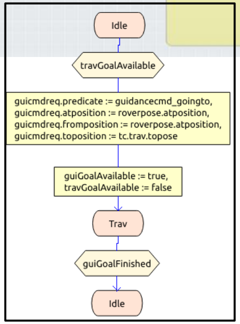
Data view

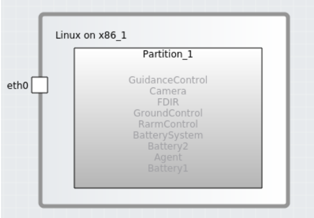
```
Position ::= Vector3d

Pose2D ::= SEQUENCE {
    pos      Position2D,
    orient   T-Double
}

-- Definition of Agent-Functional types

CameraPred ::= ENUMERATED { camera-idle, camera-takingpicture, camera-fault, camera-cancel }
ScientificcameraPred ::= ENUMERATED { scientificcamera-idle, scientificcamera-scanning, scie
BatteryPred ::= ENUMERATED { battery-set, battery-cancel }
```

Interface view



SDL behavior

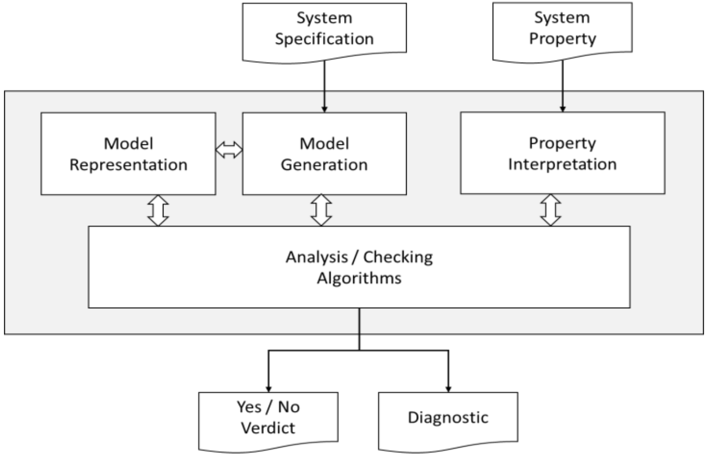




Deployment view

Model-Checking

- Automated formal verification technique for system correctness with respect to a defined set of properties
- Provides a yes/no answer for property satisfaction
- **Pros**:
  - Exhaustive exploration of the model (potentially guided by properties)
  - Fully automated
  - Easy production of diagnostic scenarios
- **Cons**:
  - State space explosion problem
  - ➢ Cannot conclude with the allocated resources (e.g., time, memory)
- **Model-checking tools**: IF, UPPAAL, NuXMV, Spin, LTSmin



Model-checking principles

Our Solution

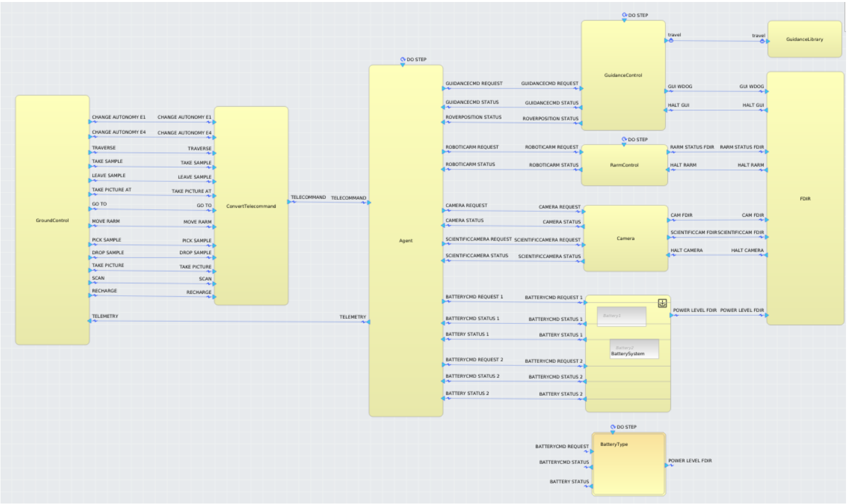
- **Aim**: Develop a model-checking technology seamlessly integrated in TASTE and validate it on representative flight software
- **Achievements**:
  - Open-source model-checking technology based on the IF toolset: https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/if/if-toolset
  - User-friendly and seamlessly integrated in TASTE, works on the modelled software, properties and defined configuration
  - Properties specified in three formalisms: simple Boolean conditions, sequences of interactions or complex state machines
  - Model-checking configuration specified through system subject to verification, possible set of inputs and model-checker options (e.g., algorithm, time limit for exploration, generation of error/success diagnostics)
  - Provides a yes, no or inconclusive result together with graphically-represented diagnostic scenarios
  - Validation on 2 case studies:
    - A subset of the Intermediate eXperimental Vehicle (IVX) on the Flaps Control System
    - An abstraction of the European Robotic Goal-Oriented (ERGO) planetary exploration demonstrator

IXV sub-orbital flight

SherpaTT in Morocco during the ERGO field tests (Courtesy of DFKI)

ERGO Case Study

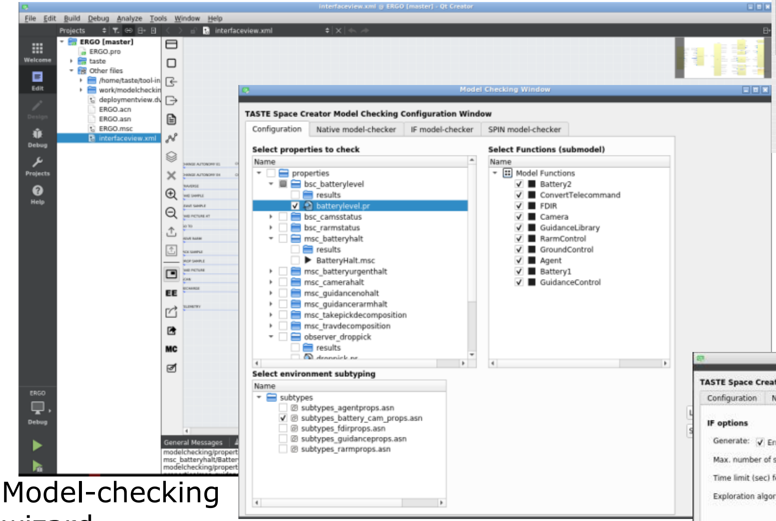
- Scenario inspired by the Mars Sample Return of an autonomous planetary exploration rover able to pick samples with a robotic arm, as well as taking images of scientific interest
- Case study consisting of the simplified functionalities of
  - Telecommanding (E1) and goal commanding (E4)
  - Simulation of traverses to specified poses, sample picking/dropping at different location, image taking of the environment (snapshots or periodically), battery operations and FDIR
- Model complexity:
  - 8 SDL functions, 1 GUI function and 1 C++ function
  - Communication through interfaces: 5 cyclic, 46 sporadic and 1 protected
- 16 properties - 3BSC, 7 MSC, 5 OBS - modelled focusing on the correct behaviour of different components:
  - 2 MSC properties for Agent
  - 1 MSC and 1 OBS for GuidanceControl
  - 1 BSC and 2 OBS for RarmControl
  - 1 BSC for Camera
  - 1 BSC for Battery1
  - 4 MSC and 2 OBS for FDIR



ERGO TASTE design subject to verification

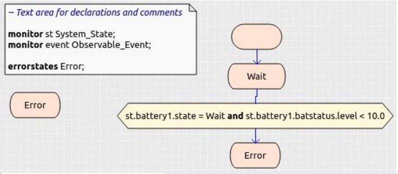
Technology Overview on ERGO

Software design in TASTE

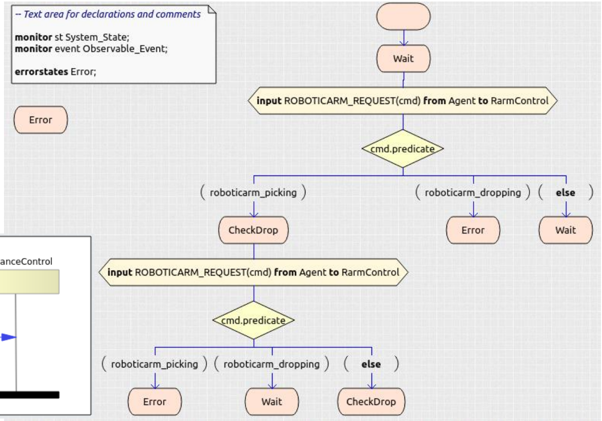


Model-checking wizard

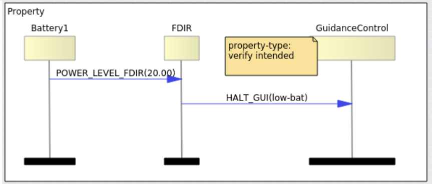
Boolean Stop Condition property

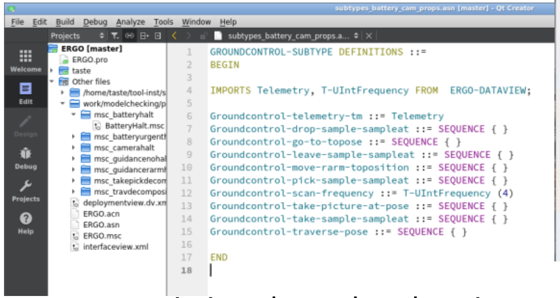


Observer property

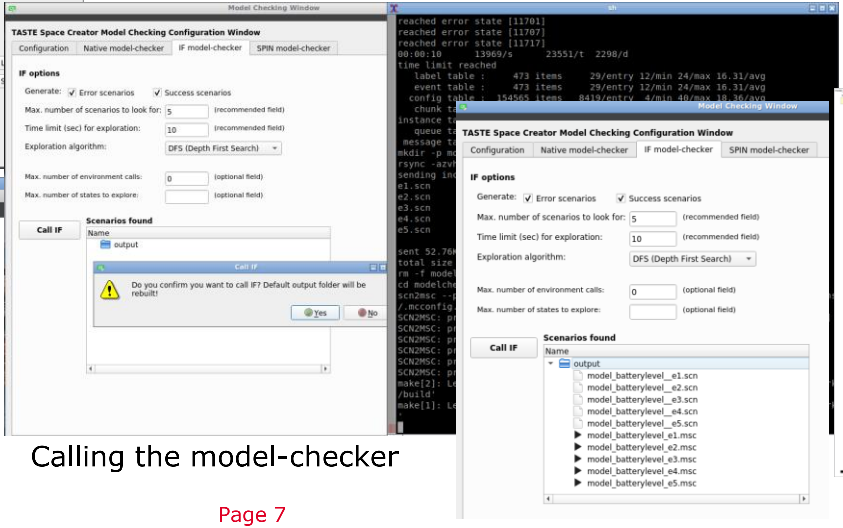


Message Sequence Chart property



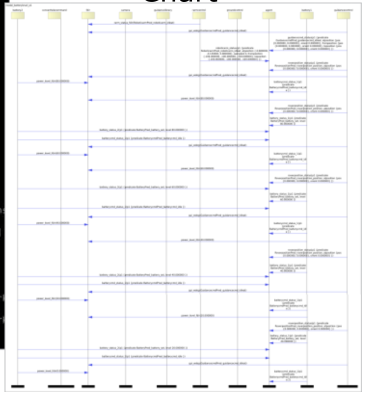


Environment restriction through subtyping



Calling the model-checker

Diagnostic scenario as Message Sequence Chart



ERGO Verification Results

Name	Type	Verif. Time	States #	Transitions #	Diagnostics # Success/Error
Low Battery1 level	BSC	3min 19sec	223679	531261	0 / 2100
RarmControl failure	BSC	17min 43sec	3974112	4338035	0 / 156
Camera failure	BSC	14sec	17646	42203	0 / 1688
Correct takepic planning	MSC – search fromstart intended	9min 03sec	1383603	2843730	48 / 0
Incorrect trav planning	MSC – search fromstart unintended	12min 45sec	2087586	4407512	0 / 2172
FDIR Camera recovery	MSC – search intended	4min 9sec	295544	586594	56 / 0
Late FDIR Battery2 level recovery	MSC – search unintended	18min 24sec	1296790	2796555	0 / 0
GuidanceControl failed halt	MSC – search unintended	2min 04sec	281145	716493	0 / 5
FDIR Battery1 recovery	MSC – verify intended	6sec	7926	13671	104 / 0
FDIR halt recovery	MSC – search intended	60min 0sec	10007572	19092513	0 / 0
FDIR watchdog	OBS	1min 40sec	240307	626117	0 / 0
Failed FDIR Camera recovery	OBS	2min 46sec	206280	466684	0 / 325
Incorrect pose achieved	OBS	22min 42sec	2923303	8894045	0 / 24
Incorrect RarmControl – drop before pick	OBS	20min16sec	3945862	43010445	0 / 1016
Incorrect RarmControl – no home position	OBS	48min58sec	2627531	2852932	0 / 51290

Average verification time: 15min

Lessons Learned (1/2)

- Assessment with respect to the following criteria:
  - Overall approach and usability of the technology
    - User-friendly technology automating most of the steps
  - System design and model-checking configuration
    - The design needs to be adapted to the verification, e.g., dedicated interfaces with the environment, data types definition easily subject to subtyping, partially support of some modelling features (C++ implementations)
  - Property specification and formalisms proposed
    - The MSC property language not expressive enough to semantically describe complex interaction properties (e.g., starting with a conjunction)
  - Identification of explicit modelling errors within the case studies
    - The MSC language not expressive enough to identify modelling errors from the diagnostic traces
  - Performance of the model-checker
    - The satisfaction of 1 property of the ERGO case study could not be concluded within 1h!

Lessons Learned (2/2)

- Guidelines for model-checking amenable system design
  - Design the software systems to enable the property projection on functions and the fine-grain control of the environment
  - Optimize the timed behavior, e.g., increase the reactivity of the system, group real-time behavior (cyclic interfaces, timers)
  - Simplify the communication between functions, e.g., upon change of status
  - Model complex interaction properties with state machines

Conclusion and Perspectives

- Step forward for large scale adoption of system design and model-checking
- Open-source technology, integrated in TASTE distribution
- Limitations:
  - Input languages: SDL, C/CPP (stateless functions), MSC/OBS; TASTE supports many more
  - State explosion is still a problem
  - Mitigated through fine-grained control of data ranges and verified functions
- Future work
  - Improve expressivity, i.e., support more constructs and/or other inputs languages, e.g., structured MSCs
  - Address the state explosion through model slicing
  - Offer more advanced simulation/debugging features

Thank you!

idragomir@gmv.com