NTNU
Norwegian University of Science and Technology

How Much Time Does Modeling Take?

Experience from Modeling Without Experience

Evelyn Honoré-Livermore

Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. Applying MBSE can be one way of achieving this. Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0 and measuring how much effort was needed. The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic. The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.

Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. Applying MBSE can be one way of achieving this. Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0 and measuring how much effort was needed. The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic. **The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years**, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.

HYPSO Project

The CubeSat and the context

Our Goal

1

Establish a pipeline for fast integration of payloads on small satellites

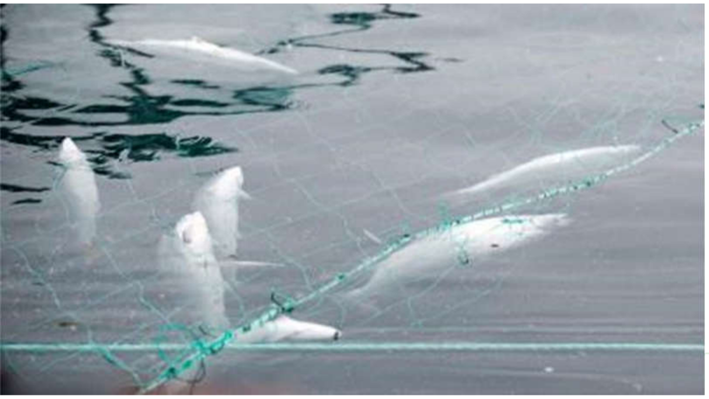
2

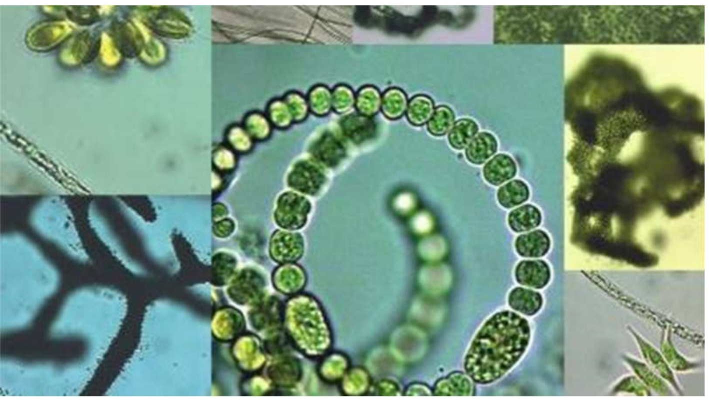
Establish a network of small satellites, drones, unmanned surface/underwater vehicles

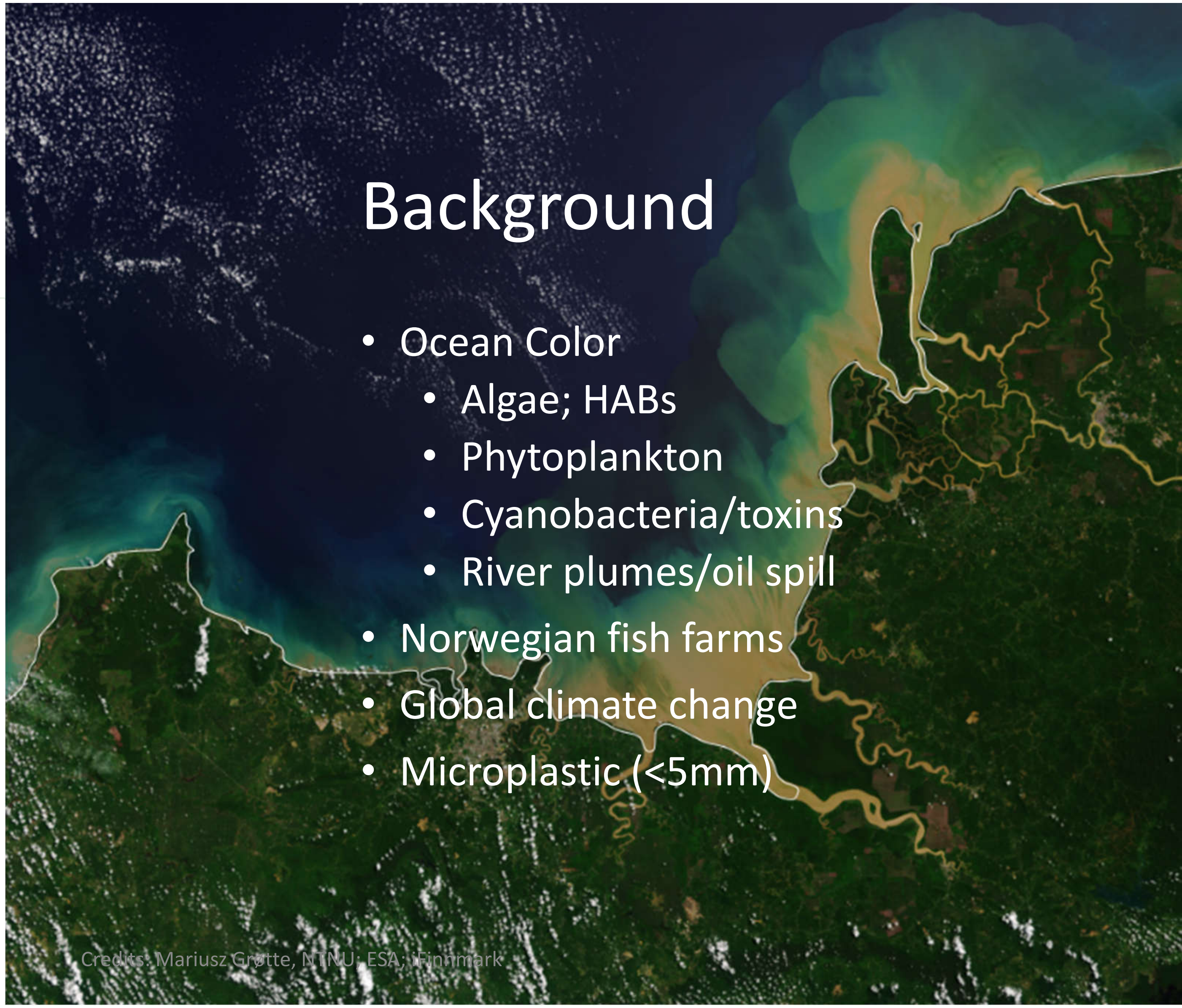
What are we trying to do?

Provide data for oceanographers

On-demand and coordinated
missions with unmanned vehicles







Background

- Ocean Color
  - Algae; HABs
  - Phytoplankton
  - Cyanobacteria/toxins
  - River plumes/oil spill
- Norwegian fish farms
- Global climate change
- Microplastic (<5mm)

Credits: Mariusz Grøtte, NTNU; ESA; iFinnmark

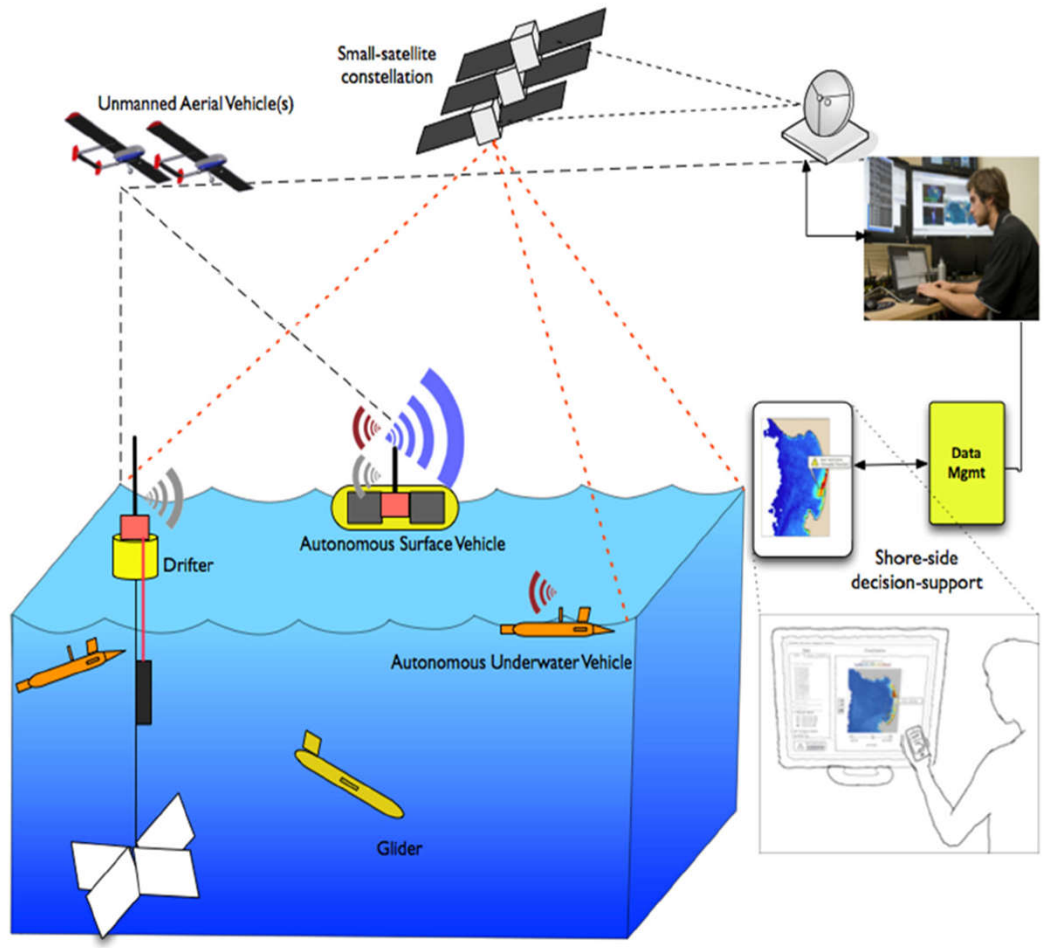
How do you get data today?

- Big satellites such as Sentinel
- Planned missions with boats and personnel that are costly and lengthy
- Manned mission in general, with drones or airplanes

- Hard to influence where and when the data is gathered

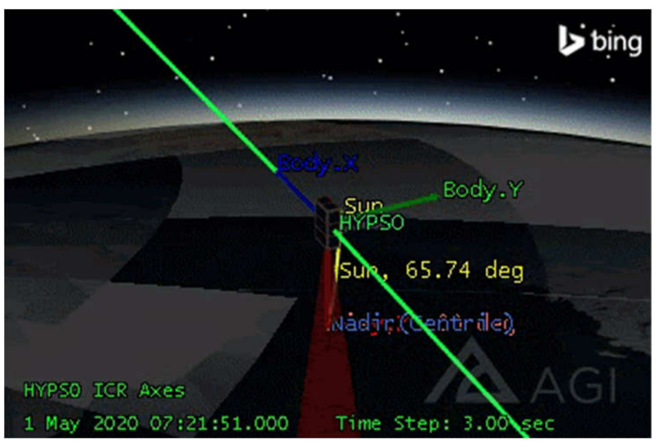
HYPSO Context

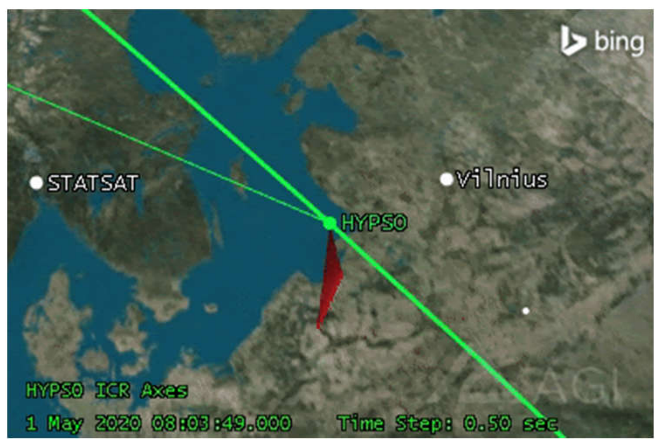
System-of-Systems

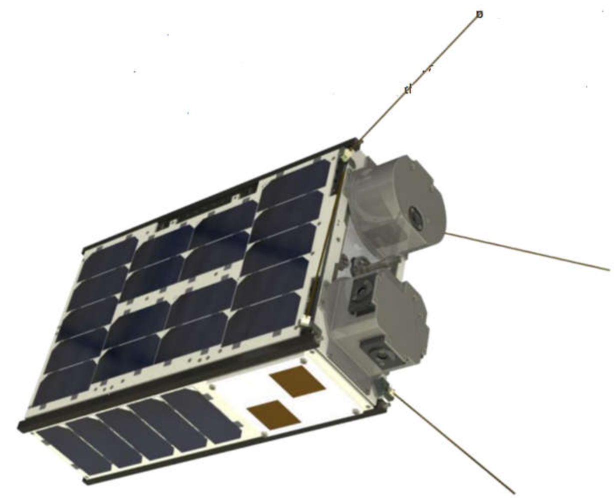


HYPSO

- 6U CubeSat at NTNU
- In-house developed hyperspectral payload
- Polar SSO orbit

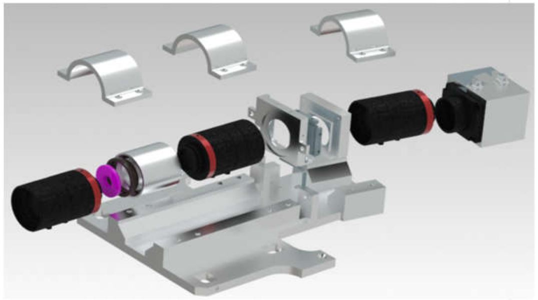


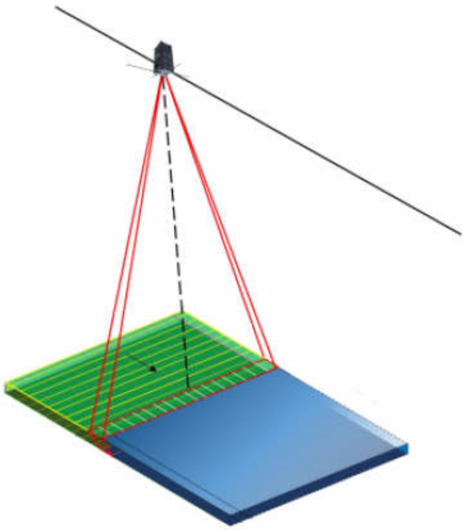


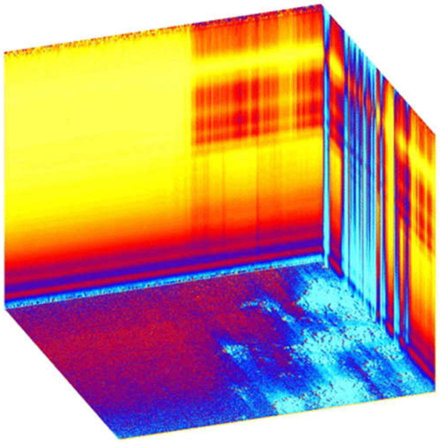


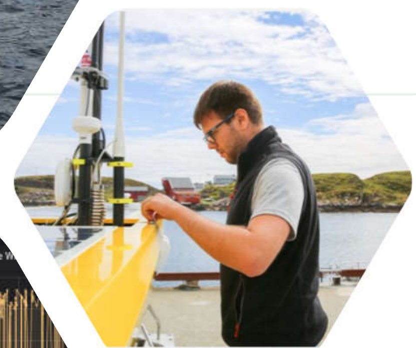
HYPSO

- Pushbroom principle
- Detect spectra



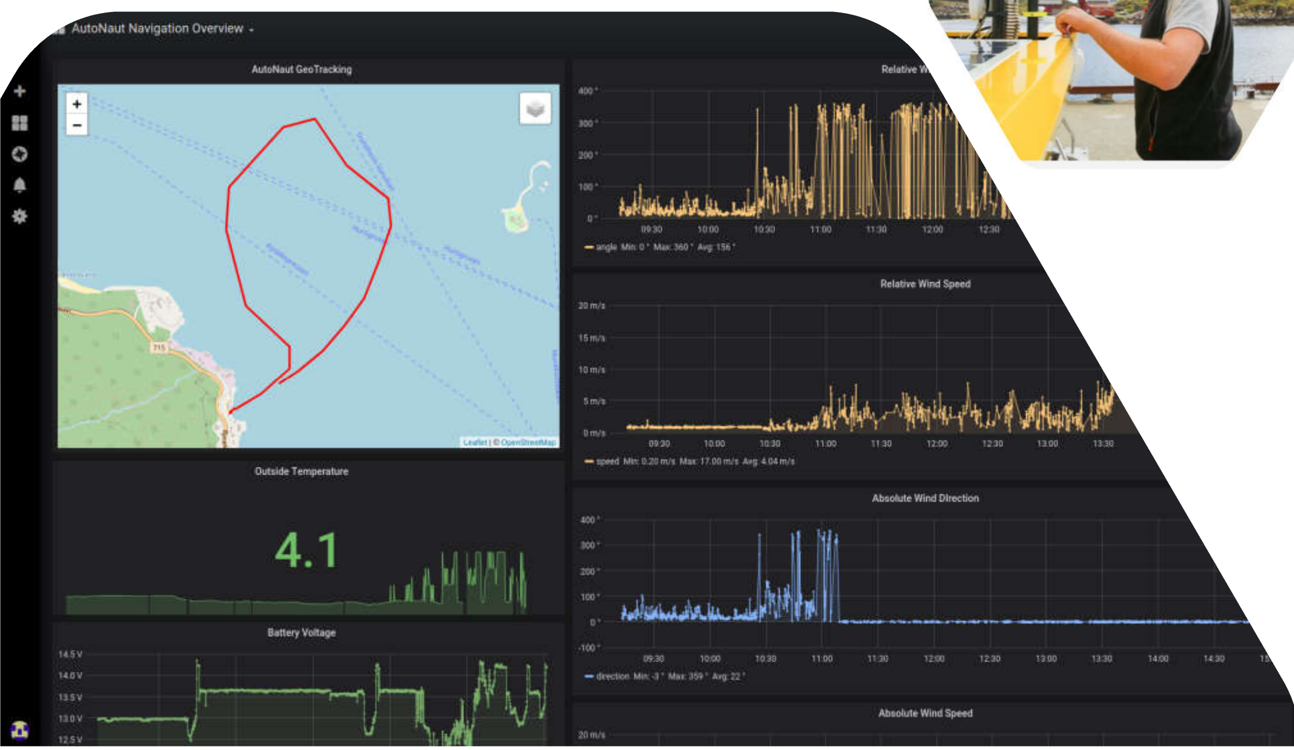






AutoNaut

Cooperative long-endurance wave powered autonomous surface vessel

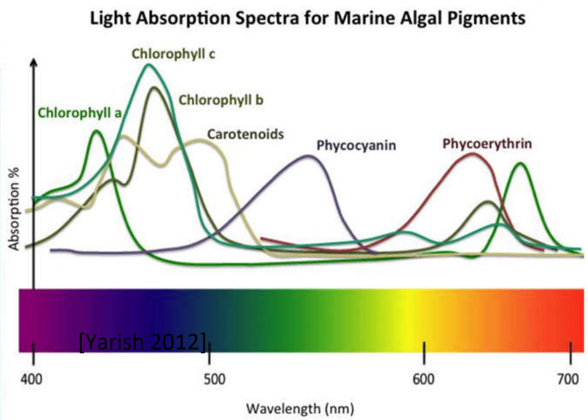


Scientific sensors

- NORTEK Signature500 ADCP
- Seabird CTD SBE 49
- Aanderaa Oxygen Optode 4835
- WET Labs ECO Puck Triplet
- Airmar 120WX Weather Station
- ThelmaBiotel TBR 700 (acoustic fish tracker)
- IMU/Sea-state (in-house, work in progress)
- Spectrometer/HSI (work in progress)

Problem: Detecting Oceanographic Phenomena

Phenomena:
- Temperature
- Salinity
- Current
- Wind
- Height
- Phytoplankton

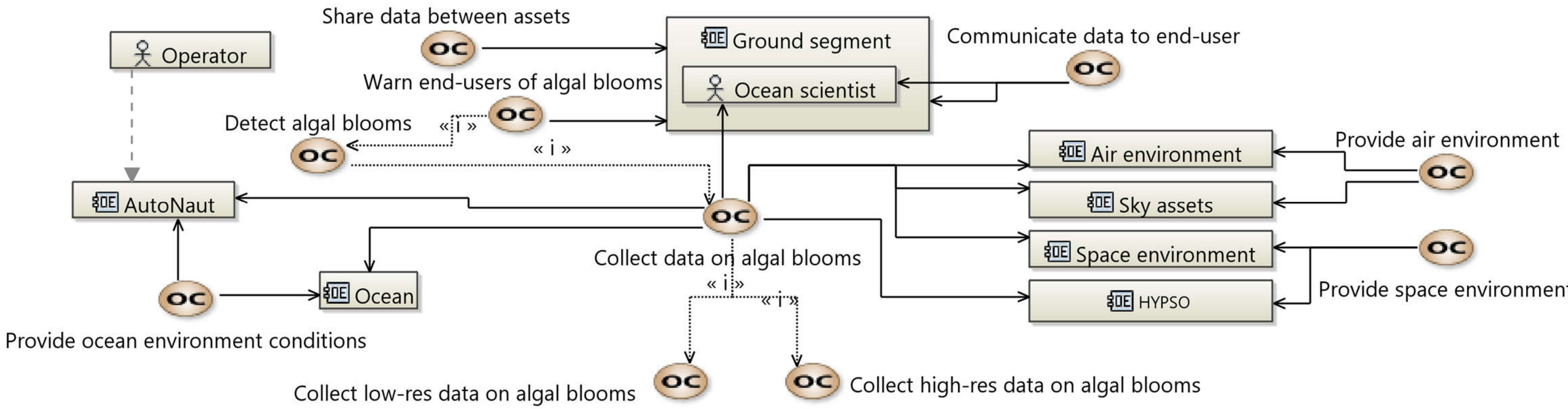


Use case: Monitoring Harmful Algal Blooms

Først fem dager etter angrepene startet, gikk «dødsalge»-alarmen

– Vi trenger bedre varslingsrutiner, slik at det blir gjort noe umiddelbart hvis alarmen går i fremtiden, sier Robert Eriksson fra Sjømatbedriftene.

Operational Analysis - Capabilities

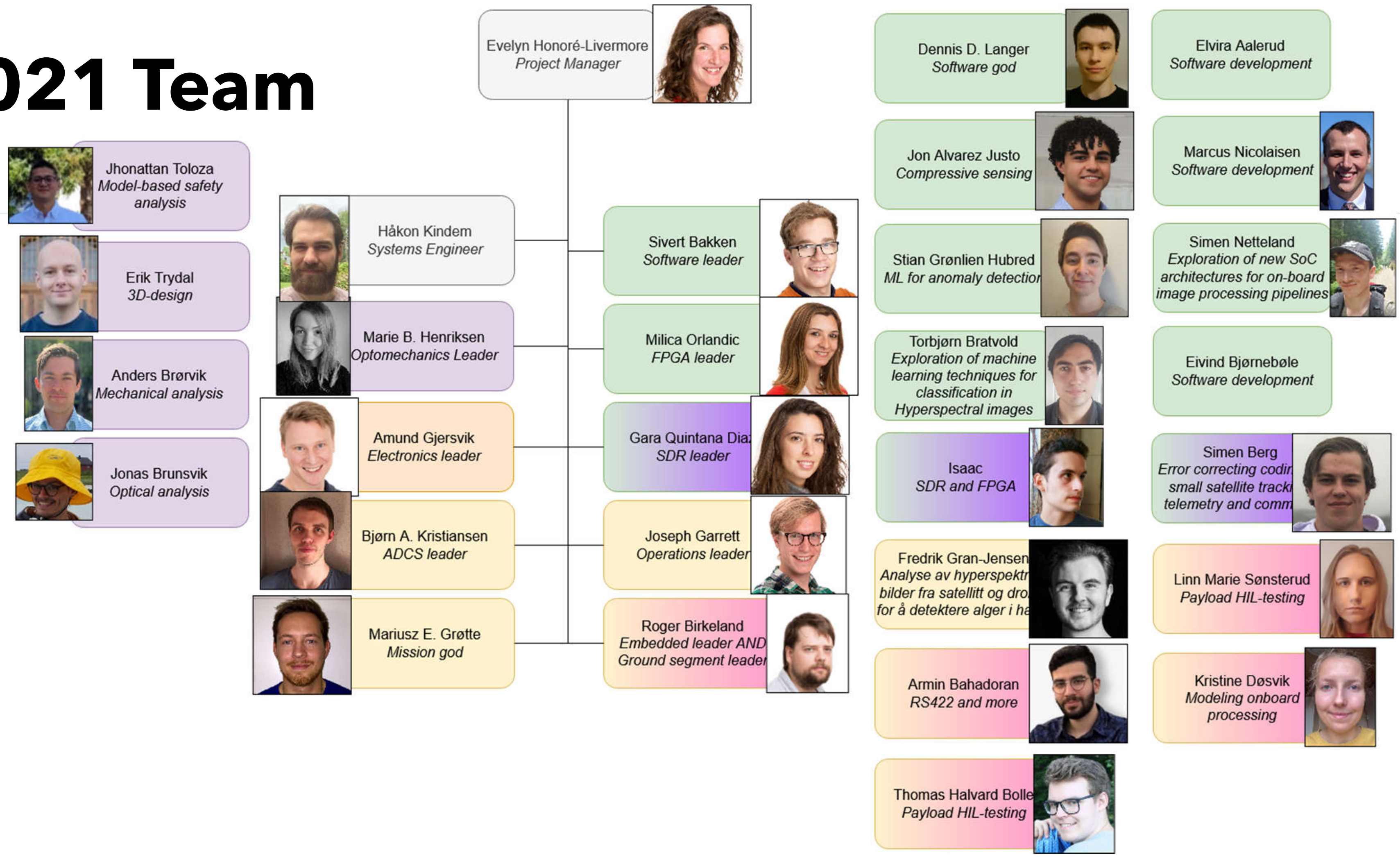


Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. Applying MBSE can be one way of achieving this. Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0, and measuring how much effort was needed. The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic. The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.

HYPSO team

2021 Team



Evelyn Honoré-Livermore
Project Manager
Jhonattan Toloza
Model-based safety analysis
Erik Trydal
3D-design
Anders Brørvik
Mechanical analysis
Jonas Brunsvik
Optical analysis
Håkon Kindem
Systems Engineer
Marie B. Henriksen
Optomechanics Leader
Amund Gjersvik
Electronics leader
Bjørn A. Kristiansen
ADCS leader
Mariusz E. Grøtte
Mission god
Sivert Bakken
Software leader
Milica Orlandic
FPGA leader
Gara Quintana Dia
SDR leader
Joseph Garrett
Operations leader
Roger Birkeland
Embedded leader AND
Ground segment leader
Dennis D. Langer
Software god
Jon Alvarez Justo
Compressive sensing
Stian Grønlien Hubred
ML for anomaly detectio
Torbjørn Bratvold
Exploration of machine learning techniques for classification in Hyperspectral images
Isaac
SDR and FPGA
Fredrik Gran-Jensen
Analyse av hyperspekt
bilder fra satellitt og dro
for å detektere alger i ha
Armin Bahadoran
RS422 and more
Thomas Halvard Bolle
Payload HIL-testing
Elvira Aalerud
Software development
Marcus Nicolaisen
Software development
Simen Netteland
Exploration of new SoC architectures for on-board image processing pipelines
Eivind Bjørnebøle
Software development
Simen Berg
Error correcting codi
small satellite track
telemetry and comm
Linn Marie Sønsterud
Payload HIL-testing
Kristine Døsvik
Modeling onboard processing

HYPSO team

- 8 PhD/PostDoc researchers
- On average 15 MSc and 6 BSc students each year
- Students join for their bachelor and master thesis work
- Students join in late August and finish in May

Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. **Applying MBSE can be one way of achieving this.** Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0 and measuring how much effort was needed. The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic. The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.

Model-Based Systems Engineering goals for HYPSO project

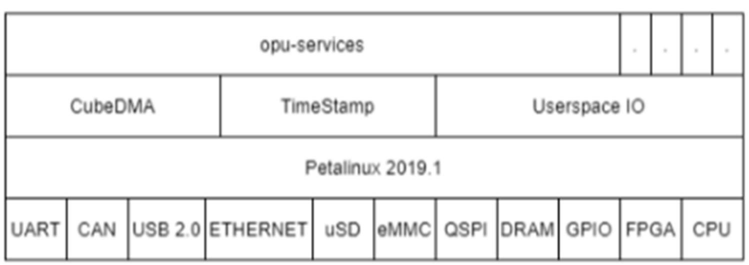
Improve	Improve knowledge management
Reduce	Reduce onboarding time of new team members
Better	Better traceability
Exchange	Efficiently exchange data between project groups
Reuse	Reuse of model elements across projects and assets

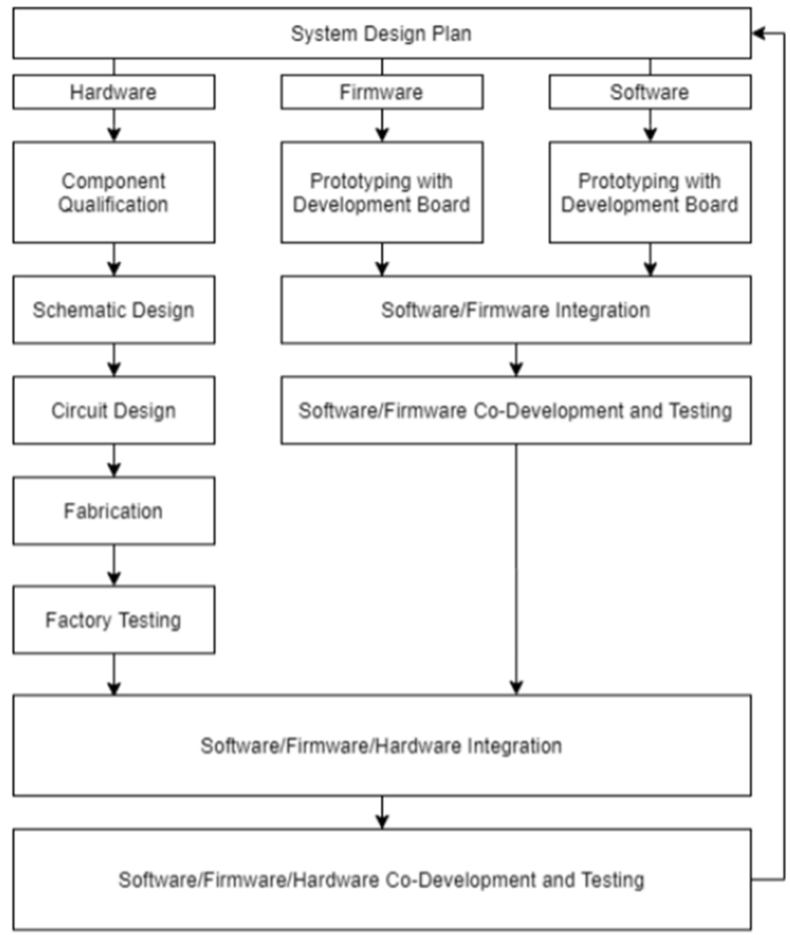
Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. Applying MBSE can be one way of achieving this. **Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0, and measuring how much effort was needed.** The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic. The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.

The Bootloader

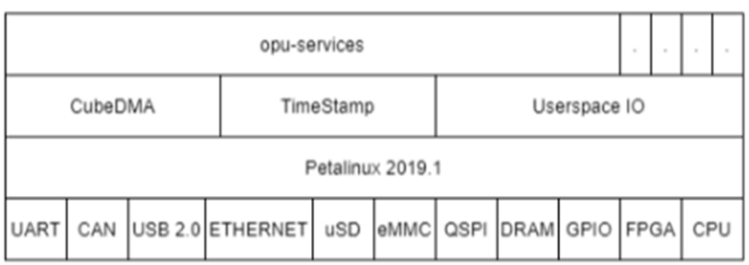
- Hardware-firmware-software co-design flow

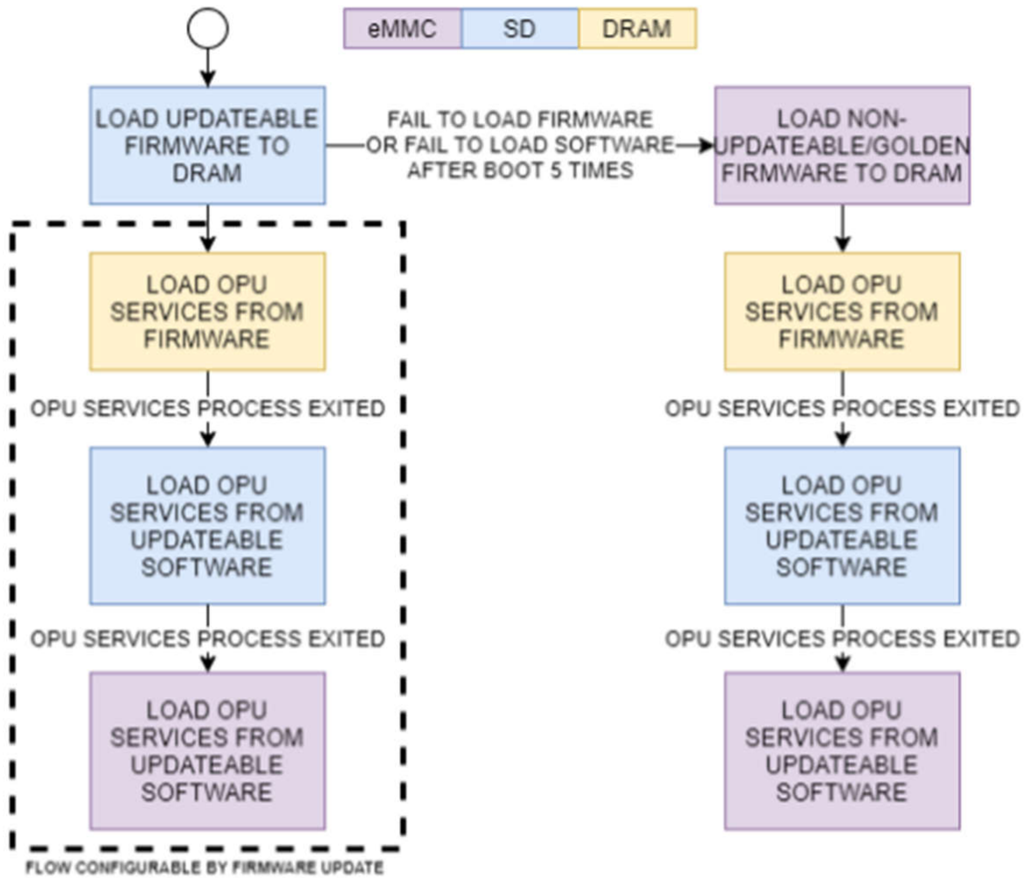




The Bootloader

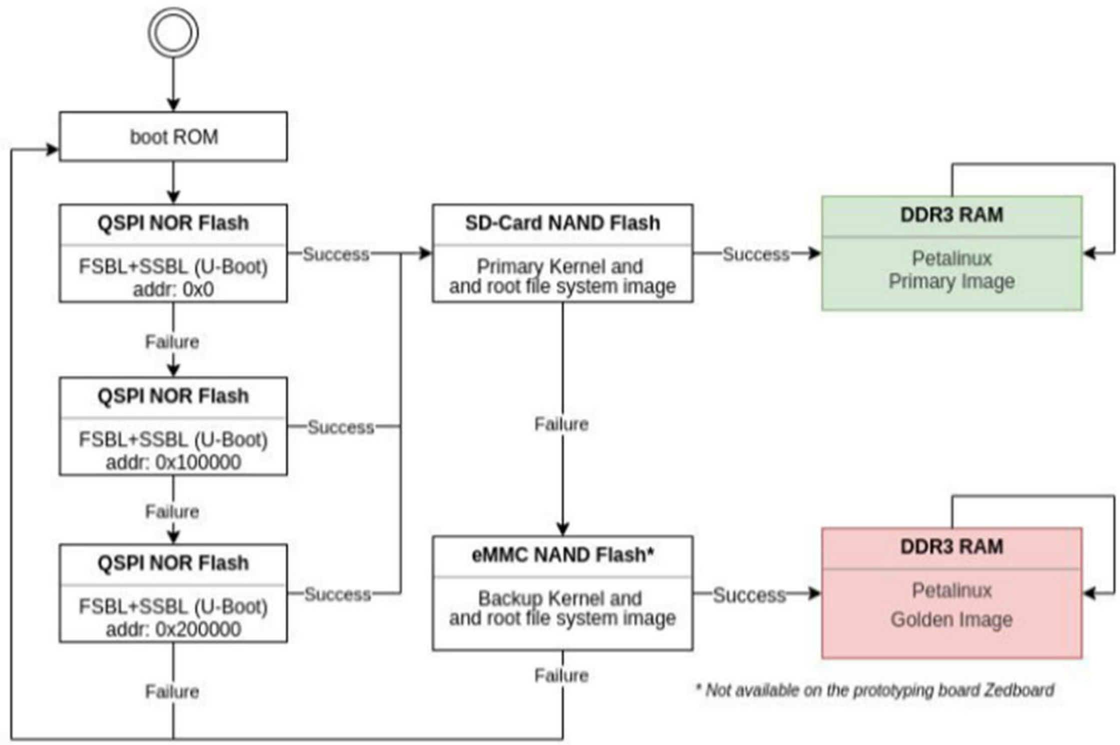
- Hardware-firmware-software co-design flow
- Firmware and software execution flow





The Bootloader

- Hardware-firmware-software co-design flow
- Firmware and software execution flow
- Final booting procedure



The modeler

- Background in mechanical and systems engineering
- No experience with MBSE previously
- Some experience with coding from previous courses
- Did the Capella catapult tutorial and watched some webinars before starting the modeling effort

Modeling process

Learn Capella

- Do tutorial and watch webinars

Software functionality

- Meeting with software responsible, project manager
- Choose function to model

Understand bootloader

- Limitations and functionality
- Intended vs. implemented design

Model bootloader

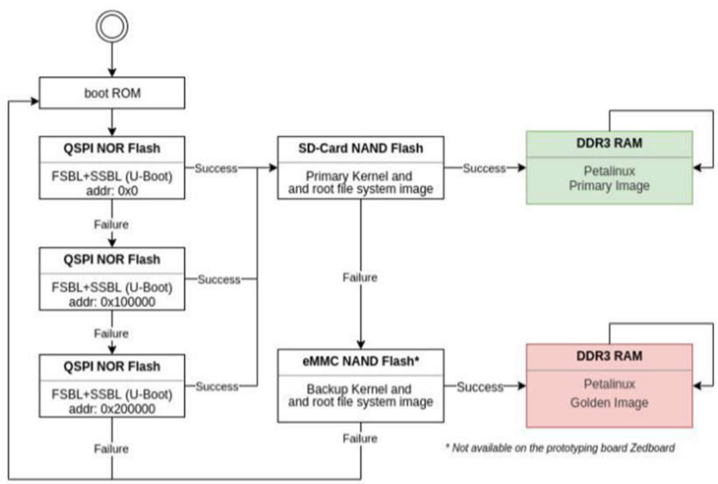
- Review with supervisors and group leaders
- New model iteration

Future work

- Use model to simplify code
- Look for redundancies
- Identify dependability issues

Bootloader modeling I

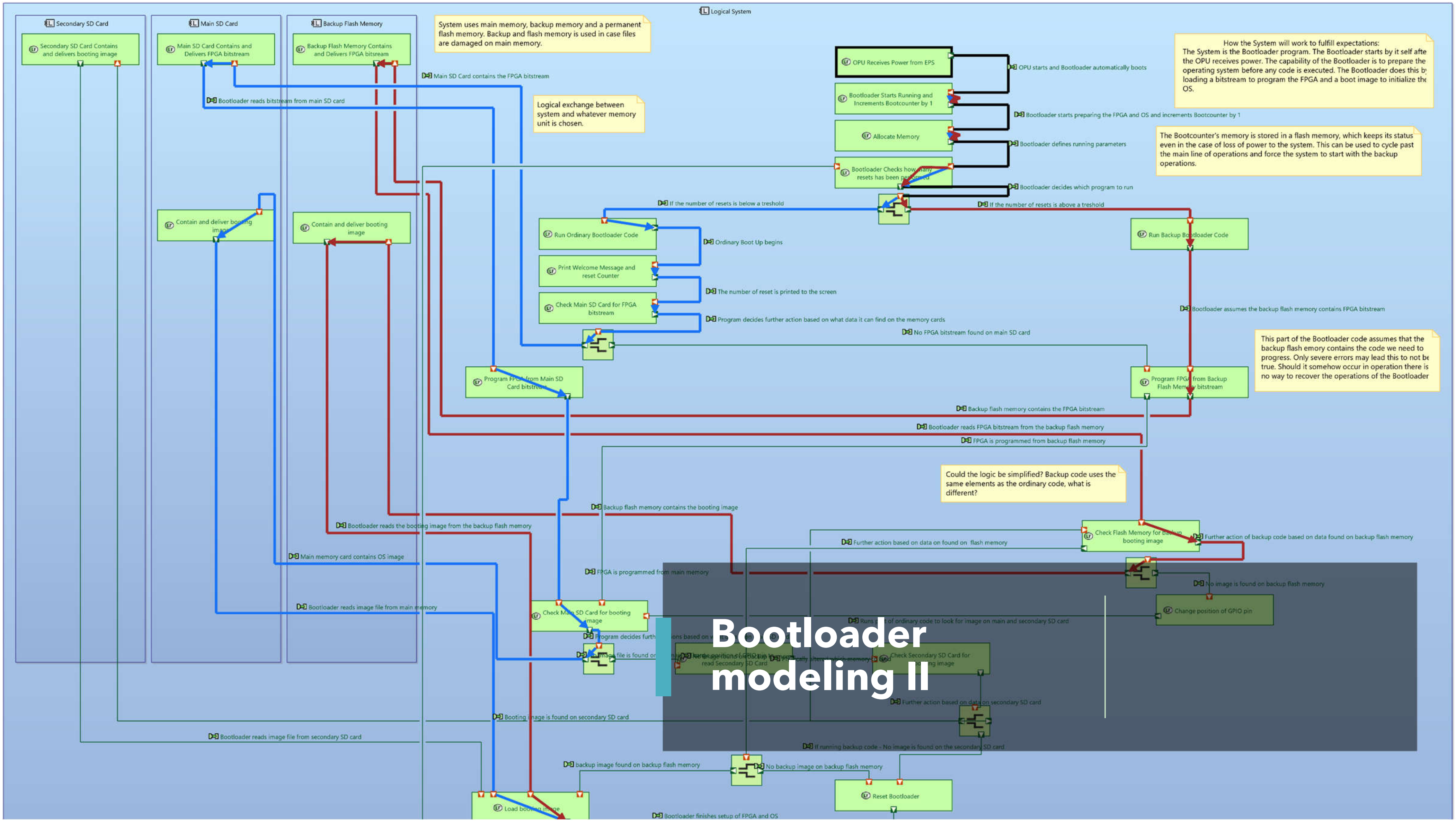
- 216 lines of code
- Very sequential

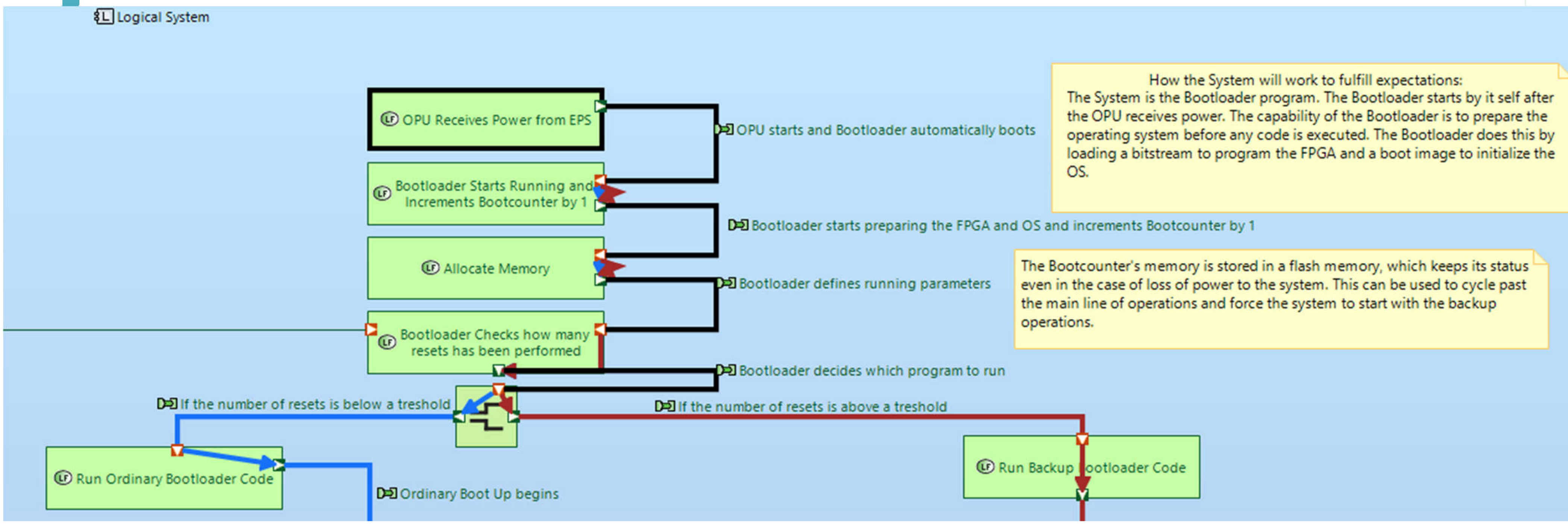


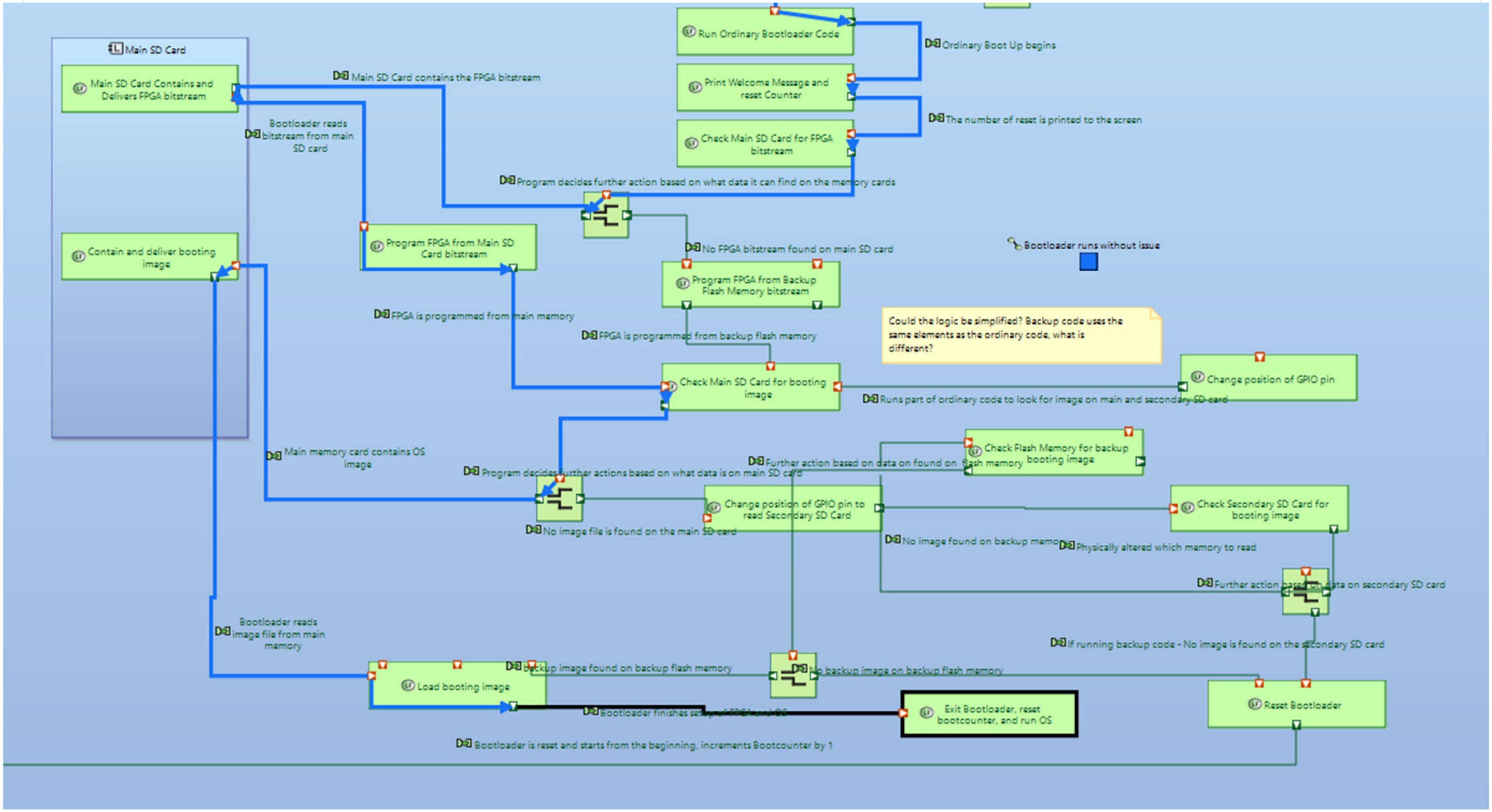
Executable File | 216 lines (164 sloc) | 10.3 KB

```
#!/bin/bash

#-----------------------------------------------------------------------------------------------------------#
#clean everything

# MEMORY CONFIGURATIONS:
# Variable name              location                                                    Description
# CONFIG_SYS_TEXT_BASE       meta-plnx-generated/recipes-bsp/u-boot/configs/config.cfg   ?
#
#
#
#
#


rm -rf pico-primary
rm -rf pico-golden
rm -rf zed-primary
rm -rf zed-golden
########################################################## GLOBAL SETTINGS ######################################

###############################################START BOOT 1#####################################################
# PRIMARY SYSTEM IMAGE (GREEN) and bootloader
main_name=${1}
project_name="${1}-primary"


load_bsp "$@"
cd $project_name

#Kernel Configs:

#set host name
sed -i 's/CONFIG_SUBSYSTEM_HOSTNAME/#/g' project-spec/configs/config

  echo 'SRC_URI += "file://kernel.cfg"' >> project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend
  echo 'FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"' >> project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend

  petalinux-config -c kernel --silentconfig
fi

# UBOOT BOOT_QSPI.BIN BOOT COMMANDS:
bootcmd_init='#define CONFIG_BOOTCOMMAND "'
bootcmd_welcomeText="echo HYPSO-1 Booting. Current bootcount is \$bootcount of \$bootlimit ; "
bootcmd_bootSD="echo booting from SD; fatload mmc 0 \$loadaddr image.ub; bootm \$loadaddr;"
bootcmd_bootSD_golden="echo booting from SD; fatload mmc 0 \$loadaddr image_golden.ub; bootm \$loadaddr;"
bootcmd_bootEMMC="echo Booting from eMMC; fatload mmc 1 \$loadaddr image_golden.ub; bootm \$loadaddr;"
bootcmd_programPL_sdfirst="if fatload mmc 0 \$loadaddr bitstream.bit notfound; then fpga loadb 0 \$loadaddr \$filesize; echo Loaded FPGA from SD; else fatload mmc 1
bootcmd_programPL_emmcfirst="fatload mmc 1 \$loadaddr bitstream.bit; fpga loadb 0 \$loadaddr \$filesize; echo Altboot. Loaded FPGA from emmc. "
bootcmd_programPL_zed="fatload mmc 0 \$loadaddr bitstream.bit; fpga loadb 0 \$loadaddr \$filesize;"
bootcmd_toggleSD="echo toggling SD card; gpio toggle 46;"

bootcmd_exit=' reset; "'

bootcmd_pico=${bootcmd_init}${bootcmd_welcomeText}${bootcmd_programPL_sdfirst}${bootcmd_bootSD}${bootcmd_toggleSD}${bootcmd_bootSD}${bootcmd_bootEMMC}${bootcmd_exit}
bootcmd_zed=${bootcmd_init}${bootcmd_welcomeText}${bootcmd_programPL_zed}${bootcmd_bootSD}${bootcmd_toggleSD}${bootcmd_bootSD}${bootcmd_bootEMMC}${bootcmd_exit}

altbootcmd_pico="${bootcmd_programPL_emmcfirst}${bootcmd_bootEMMC}${bootcmd_toggleSD}${bootcmd_bootSD}${bootcmd_toggleSD}${bootcmd_bootSD} reset;"
altbootcmd_zed="${bootcmd_programPL_zed}${bootcmd_bootSD_golden}${bootcmd_bootSD} reset;"




if echo $* | grep -e "pico" -q
then
  #set alternative boot sequence for when bootcounter > 5. bootm size defines max size for image in bytes (0x10000000=256 MB)
  sed -i 's#ENV_SETTINGS#ENV_SETTINGS "upgrade_available=1\\0" \\ \n "altbootcmd='"$altbootcmd_pico"'\\0" \\ \n "bootm_size=0x10000000\\0" \\ \n #g' project-spec/met
  # needed because we want bootm size variable to define max image size.
  echo "#undef CONFIG_SYS_BOOTMAPSZ" >> project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h

  echo  ${bootcmd_pico} >> project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h

else
  #set alternative boot sequence for when bootcounter > 5. bootm size defines max size for image in bytes (0x10000000=256 MB)
  sed -i 's#ENV_SETTINGS#ENV_SETTINGS "upgrade_available=1\\0" \\ \n "altbootcmd='"$altbootcmd_zed"'\\0" \\ \n  "bootm_size=0x10000000 \\0" \\ \n #g' project-spec/me
  echo  ${bootcmd_zed} >> project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h
fi
```

Bootloader modeling II



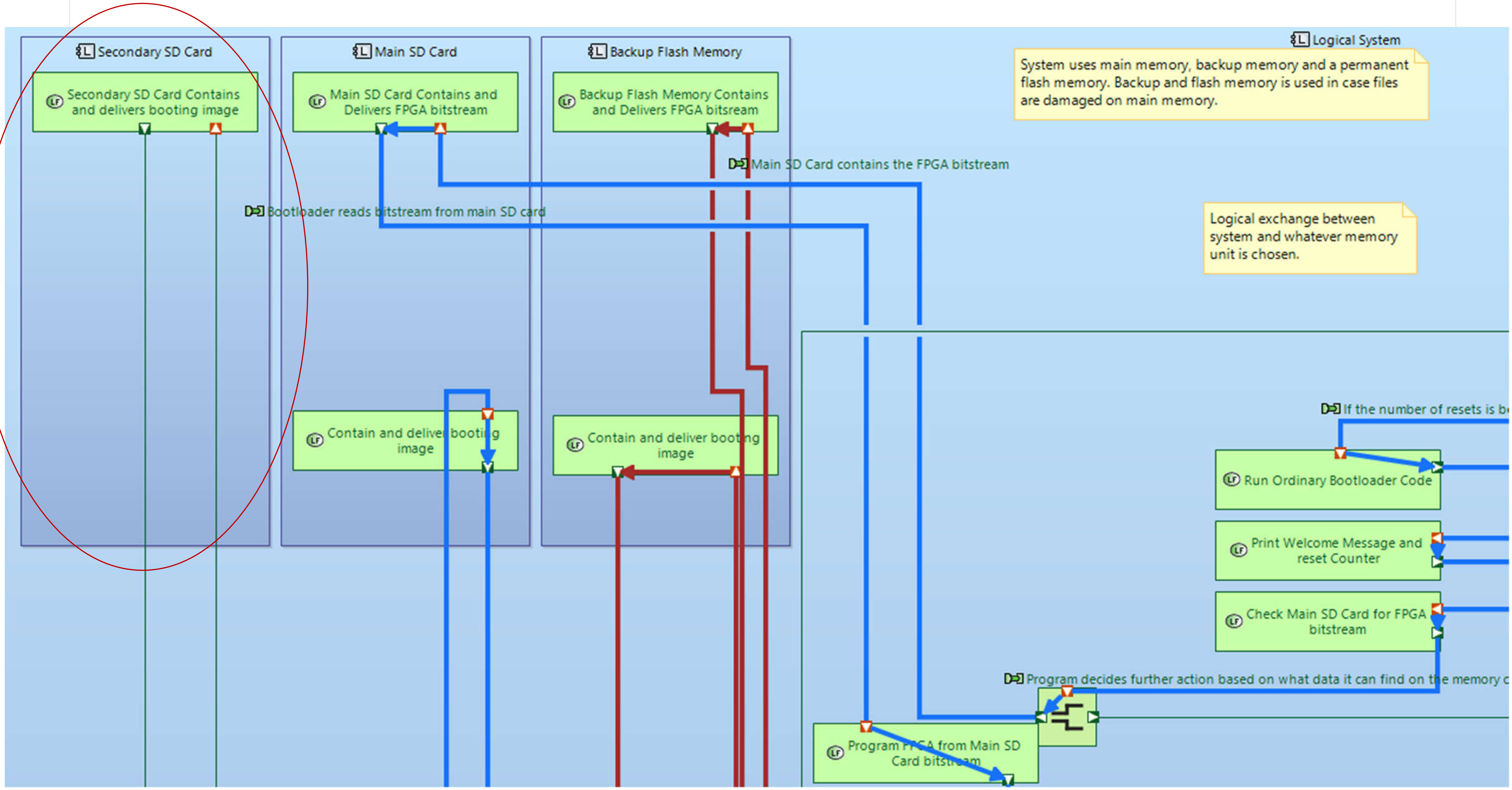

Logical System
OPU Receives Power from EPS
OPU starts and Bootloader automatically boots
Bootloader Starts Running and Increments Bootcounter by 1
Bootloader starts preparing the FPGA and OS and increments Bootcounter by 1
Allocate Memory
Bootloader defines running parameters
Bootloader Checks how many resets has been performed
Bootloader decides which program to run
If the number of resets is below a treshold
If the number of resets is above a treshold
Run Ordinary Bootloader Code
Ordinary Boot Up begins
Run Backup Bootloader Code
How the System will work to fulfill expectations:
The System is the Bootloader program. The Bootloader starts by it self after the OPU receives power. The capability of the Bootloader is to prepare the operating system before any code is executed. The Bootloader does this by loading a bitstream to program the FPGA and a boot image to initialize the OS.
The Bootcounter's memory is stored in a flash memory, which keeps its status even in the case of loss of power to the system. This can be used to cycle past the main line of operations and force the system to start with the backup operations.

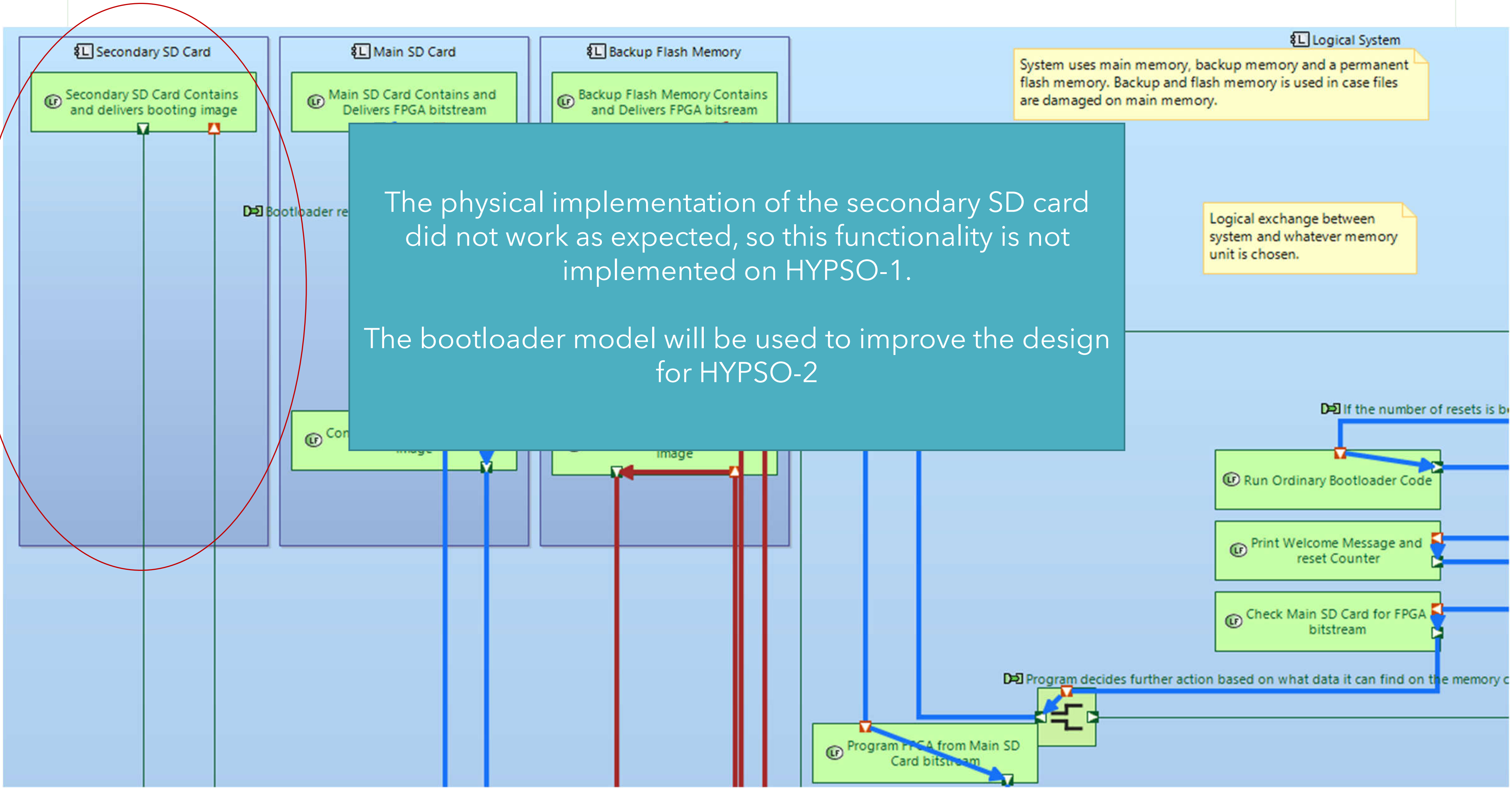


Main SD Card
Main SD Card Contains and Delivers FPGA bitstream
Contain and deliver booting image
Main SD Card contains the FPGA bitstream
Bootloader reads bitstream from main SD card
Run Ordinary Bootloader Code
Ordinary Boot Up begins
Print Welcome Message and reset Counter
The number of reset is printed to the screen
Check Main SD Card for FPGA bitstream
Program decides further action based on what data it can find on the memory cards
Program FPGA from Main SD Card bitstream
No FPGA bitstream found on main SD card
Bootloader runs without issue
Program FPGA from Backup Flash Memory bitstream
FPGA is programmed from main memory
FPGA is programmed from backup flash memory
Could the logic be simplified? Backup code uses the same elements as the ordinary code, what is different?
Check Main SD Card for booting image
Change position of GPIO pin
Runs part of ordinary code to look for image on main and secondary SD card
Check Flash Memory for backup booting image
Main memory card contains OS image
Further action based on data on found on flash memory
Program decides further actions based on what data is on main SD card
Change position of GPIO pin to read Secondary SD Card
Check Secondary SD Card for booting image
No image file is found on the main SD card
No image found on backup memory
Physically altered which memory to read
Further action based on data on secondary SD card
Bootloader reads image file from main memory
If running backup code - No image is found on the secondary SD card
Load booting image
Backup image found on backup flash memory
No backup image on backup flash memory
Reset Bootloader
Bootloader finishes setup of FPGA and OS
Exit Bootloader, reset bootcounter, and run OS
Bootloader is reset and starts from the beginning, increments Bootcounter by 1

Abstract

A shared common mental model of a system design between team members is a goal many projects aspire to. Applying MBSE can be one way of achieving this. Here we present the results of an inexperienced modeler taking existing code logic and modeling it in Capella 5.0, and measuring how much effort was needed. **The models make the logic of the program more accessible to coders who did not work on that specific piece of code previously, which can increase the possibility of code review and improving logic.** The case study was a University CubeSat team, where team members join the project as a part of their thesis, while the project continues for 2-3 years, and modeling the code logic can reduce some of the onboarding effort required when new members want to reuse or improve on existing codebase.


Secondary SD Card
Secondary SD Card Contains and delivers booting image
Main SD Card
Main SD Card Contains and Delivers FPGA bitstream
Backup Flash Memory
Backup Flash Memory Contains and Delivers FPGA bitsream
Logical System
System uses main memory, backup memory and a permanent flash memory. Backup and flash memory is used in case files are damaged on main memory.
Main SD Card contains the FPGA bitstream
Bootloader reads bitstream from main SD card
Logical exchange between system and whatever memory unit is chosen.
Contain and deliver booting image
Contain and deliver booting image
If the number of resets is b
Run Ordinary Bootloader Code
Print Welcome Message and reset Counter
Check Main SD Card for FPGA bitstream
Program decides further action based on what data it can find on the memory c
Program FPGA from Main SD Card bitstream



The physical implementation of the secondary SD card did not work as expected, so this functionality is not implemented on HYPSO-1.

The bootloader model will be used to improve the design for HYPSO-2

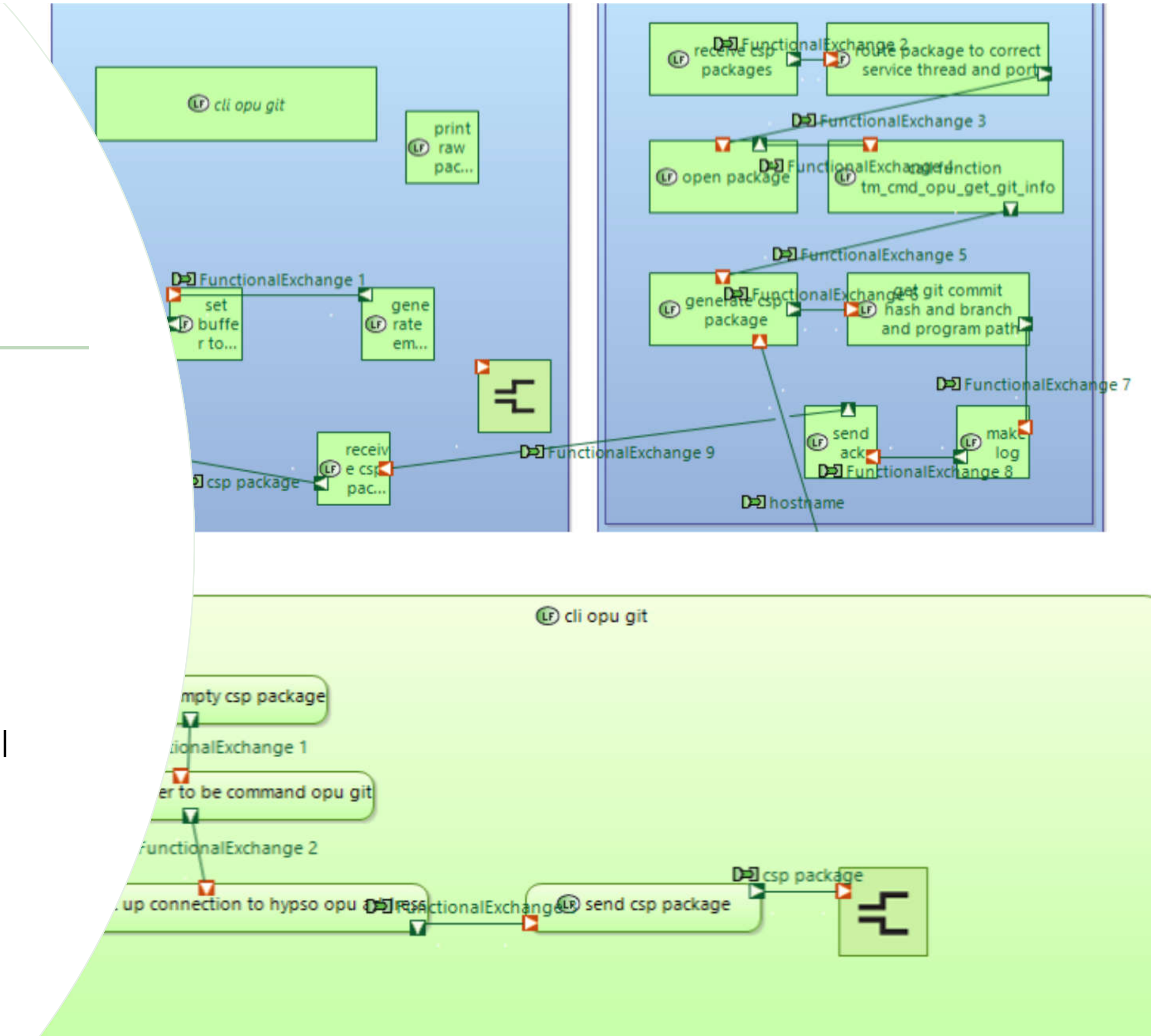
```c
int cli_opu_git(char* args)
{
    (void)args;
    int packet_length = 1;
    csp_packet_t* packet = csp_buffer_get(packet_length);

    packet->data[0] = OPU_CMD_GETGIT;
    packet->length = packet_length;

    csp_conn_t* conn;
    conn = csp_connect(0, HYPSO_OPU_ADDRESS, OPU_TM_PORT, 1000, CSP_O_NONE
    print_raw_packet(HYPSO_OPU_ADDRESS, OPU_TM_PORT, packet);
    if (csp_send(conn, packet, 1000) != 1)
    {
        printf("    Could not send request.\n");
        csp_buffer_free(packet);
        csp_close(conn);
        return ECOMM;
    }

    // Wait for ACK
    csp_packet_t* return_packet = csp_read(conn, 3000);
    if (return_packet == NULL)
    {
        printf("    ACK Timeout.\n");
        csp_close(conn);
        return ETIMEDOUT;
    }
    return_packet->data[return_packet->length] = 0;
    printf("<-- %s", return_packet->data);
    csp_buffer_free(return_packet);
    csp_close(conn);

    return 0;
}
```

How much time does it take?

- 162 lines of code = 25 hours including time spent learning Capella
- So, we tried with another modeler and another functionality to see if we could get more data

cli opu git command

- Low-level command
- No extra documentation provided
- Only code lines
- Difficult to model without understand code
- Gave up and decided to go higher level instead after spending 3 hours trying to model function of 34 lines of code



Lessons learned

- Academic CubeSat teams are limited in Systems Engineering resources and experience – cannot choose or control who joins the project at any time
- Need to find ways to do SE activities without too large overhead
- Capella is open source and has a large online community which facilitates learning
- Can collaborate on model through Git
- Challenging to train engineers in MBSE in addition to their discipline (radio comm., ADCS, SW dev.) and develop understanding of what is needed to contribute to MBSE process
- **The first functionality to model should be higher-level code rather than low-level, where more knowledge of software systems is needed**

Way forward

- Model HYPSO-2 from top-down
- And bottom-up from existing functionality re-used from HYPSO-1
- Use for identifying functions not implemented to give issues in GitHub for students
- And dependability analysis
- And verification and validation (hopefully)

Norwegian University of Science and Technology

Thank you

- Flow diagrams from Joar Gjersund's master thesis
- Bootloader model from Runar G Rovik's master thesis
- Work from Runar G Rovik's master thesis