Considering Abstraction Levels on a Case Study

Casper Thule and René Nilsson

Aarhus University, Department of Engineering
Finlandsgade 22, 8200 Aarhus N, Denmark
{casper.thule,rn}@eng.au.dk

Abstract. Cyber-physical systems consist of cyber parts controlling physical entities and this close interaction can be challenging. To manage the complexity of CPSs it can be useful to create models of the constituent components and simulate these in what is called a co-simulation. This can help in building prototypes and discovering undesired behaviour. When creating such models it is important to choose the right abstraction level to enable prototyping of various parts of a system. For this purpose it can be advantageous to create models at different levels of abstraction. This paper describes a case study based on a continuous time model and a discrete event model of a quadrotor unmanned aerial vehicle. Abstractions are considered a tool to gain insights and manage the complexity of a given system, and therefore the discrete event model has been abstracted to allow focusing on high-level waypoint behaviour. The results show that the abstracted models resemble the original model with respect to the goals of the abstraction.

Keywords: Cyber-Physical Systems, Co-Simulation, Model-Based Design, Crescendo, Overture, VDM-RT, DESTECS, INTO-CPS, FMI

1 Introduction

Cyber-Physical Systems (CPSs) incorporate sensing, actuating, computing, and communication from a cyber perspective [14]. Such systems are becoming a vital part of our society, which relies on them in cars, trains, medical devices, and so forth. The development of CPSs poses a challenge because of the close interaction between cyber parts and physical entities. This challenge is also due to the necessity of taking the complexity of the physical domain into account when developing control applications [12].

To support the development of CPSs it can be useful to create models of the constituent components that jointly form a given system. A model is an abstract description of a component, where irrelevant details are abstracted away [10,16]. In case these components are expressed in an executable subset of a notation, a collaborative simulation (a co-simulation) can be employed, which can couple models created in different formalisms. One way of performing such co-simulations is presented in the DESTECS[1] project, where Continuous Time

[1] http://www.destecs.org/, visited on 28 Oct. 2016.

(CT) and Discrete Event (DE) models can be co-simulated. The approach of using models is captured in "model-based design", a methodology that describes how models can be used when developing new systems [9]. In this paper the physical system dynamics are modeled in the CT modeling domain and the digital control in the DE modeling domain [5]. The model-based design approach excels in managing complexity [4], which is an important attribute as it is necessary to allow for increasing complexity of CPSs [8].

This paper concerns reflecting on the chosen level of abstraction used in modeling the DE side of a quadrotor Unmanned Aerial Vehicle (UAV), shown in Fig. 1, in a case study [6]. Furthermore, it demonstrates how two abstractions of the DE model can serve for prototyping purposes in a learning process. The collaborative model of the UAV consists of a CT model representing the physical system dynamics and a DE model representing the control logic. The CT model captures the system properties that are continuous in time, e.g. physical phenomena, whereas the DE modeling domain captures system functionality that is executed at discrete time intervals, e.g. control algorithms.

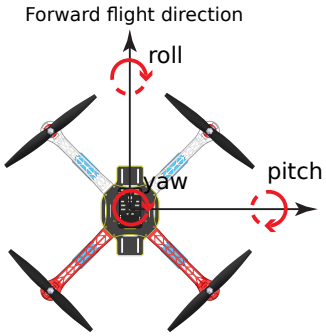

Fig. 1. The model of the UAV. Yaw, pitch, and roll are Euler angles, which are described in Sect. 3.

The Crescendo technology developed in the DESTECS project enables the possibility of collaboratively modeling and simulating the UAV. The CT parts of the system are modeled in 20-sim using differential equations, block diagrams, and bond graphs [2]. Discrete events are modeled in Overture using VDM-RT [11,13]. The coupling is achieved through the Crescendo tool[2], which functions as a co-simulation engine for the CT model and the DE model. The DE and CT model for the UAV are jointly referred to as co-model. The Crescendo technology is limited in the sense that it supports a maximum of one CT and one DE model, which have been designed specifically for the Crescendo technology with proprietary interfaces.

[2] http://crescendotool.org/, visited on 28 Oct. 2016.

Using the tools developed in the INTO-CPS project [3] it is possible to perform co-simulations with models adhering to the the standardized Functional Mock-up Interface (FMI). FMI describes an interface in the C language and a model/component implementing this along with providing a model description is called a Functional Mock-up Unit (FMU). The INTO-CPS project concerns development of CPSs from requirements through design, down to realisation in hardware and software. It encompasses the tools 20-sim and Overture along with functionality for e.g. performing Design Space Exploration (DSE), verification, and co-simulation with hardware in the loop [7]. Therefore, the CT and DE models mentioned above can be represented as FMUs and co-simulated using INTO-CPS technology.

After this introduction Sect. 2 presents the development history of both the CT and DE models of the UAV model and the motivation for the work presented in this paper. Sect. 3 will then give an overview of the CT model. Next, Sect. 4 describes the DE model, which is based on an open source quadrotor control application called APM:Copter[3]. To improve the DE model and gain more insight into the development of control logic for a UAV Sect. 5 presents a case study and reflects on abstraction levels. Furthermore, a proposal for a new DE model is presented. Afterwards, Sect. 6 describes the transition from the Crescendo technology to the INTO-CPS technology. Finally, Sect. 7 presents concluding remarks on this paper and Sect. 8 outlines the future work on the co-model.

2 History

The work presented in this paper builds upon an earlier project in which a detailed co-model of a UAV was developed with the Crescendo technology [6]. The project used a CT-first approach [5], where initial work was commenced on the CT model, since the developers had little or no prior knowledge of UAV dynamics. Development of a detailed CT model was carried out using a refinement process, resulting in three CT models at different abstraction levels:

Conceptual model: A pure CT model of a "Flying box". This model was used to give a thorough understanding of forces and moments and coordinate systems in a UAV setting.

Generic component model: This model is a refinement of the conceptual model. Rather than a flying box, this model captures all components of a quadrotor UAV and provides valuable information about components and their interaction. Additionally, it identifies the interface to the DE model and enables co-simulations.

Specific device model: The specific device model is refined by measuring real system dynamics, such as motor/rotor characteristics, of a specific UAV and applying these measurements in the CT model.

Development of the DE model did not follow the same refinement process. Instead a detailed DE model was developed by reverse engineering parts of the existing

[3] https://www.dronecode.org/dronecode-software-platform, visited on 28 Oct. 2016.

open source framework APM:Copter. This lead to a comprehensive DE model with parts that were not fully understood, due to undocumented values and code in the framework. This DE model will henceforth be referred to as the original DE model.

The motivation for this paper is to demonstrate how an abstract version of the original DE model can resemble the functionality and therefore be used to gain insights concerning the control logic of a UAV. Additionally, the purpose of this paper is also to begin the development of a new DE model because of the undocumented implementation of the APM:Copter framework. This is necessary in order to thoroughly understand the parts that make up the control logic for a UAV and to create an architecture that improves the possibility of prototyping.

3 The Continuous Time Model

The CT model describes the physical dynamics of a DJI F450 Flamewheel quadrotor UAV, illustrated in Fig. 1. The UAV has a cross airframe and is propelled by four rotors rotated by four motors driven by four Electronic Speed Controllers (ESCs). Despite of four rotors, the system is inherently unstable, causing a need for a controller [1]. The controller reads inputs from various sensors and pilot guidance through a telemetry system. Using complex control algorithms, the controller is capable of keeping the UAV stable, as well as steering the UAV, by adjusting the four motor speeds individually. The control algorithms are realized in the DE side and described further in Sect. 4.

When developing UAV models, both in the DE and CT side, it is important to have a common notion of terms and their meaning concerning UAV position and orientation. These are most often described using two coordinate systems and a rotation convention. For simplicity the mathematical basis is not described here, but the main terms are described as follows:

Position: The position of a UAV refers to the spatial location. This is often expressed with latitude, longitude, and altitude.

Attitude: The attitude of a UAV refers to the orientation (also referred to as angular position). This can be expressed using Euler angles roll, pitch, and yaw. Each Euler angle is a measure of how much the UAV is rotated around one of its axes, illustrated in Fig. 1.

An overview of the components and the interface to the CT model are presented in Fig. 2.

The CT model receives motor setpoints for all motors from the control algorithm at the DE side through an interface defined in a contract. These setpoints drive the motor controllers (ESCs), which in turn drives the motors and rotors. The airframe of the UAV is represented as a rigid body that can move and rotate in space, when affected by forces from the rotors. The rigid body is described with ideal differential equations, from which both spatial and angular accelerations, velocities, and positions are extracted. These extracted values serve as input to various sensors, which can simulate noise, inaccuracies,

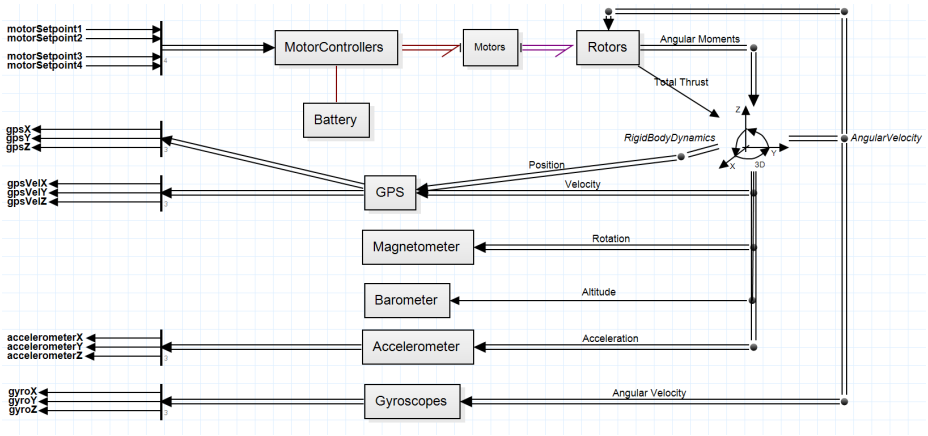

Fig. 2. CT model overview

and rounding errors. Sensor outputs are made available to the control algorithm in the DE side through the contract interface.

4 The Discrete Event Model

The original DE model was developed by reverse engineering parts of the open source framework APM:Copter, which is a control platform for multi-rotor UAVs. The architecture in the model resembles the APM:Copter project, but all irrelevant functionality is abstracted away. This includes all low level software such as operating system, drivers, and wireless communication (telemetry). Additionally, sensor fusion has been overly simplified, since ideal sensor data is available from the CT model, thus removing the sensor fusion requirement[4].

An overview of the main components of the DE model is shown in Fig. 3 whereas the components of the flight control algorithm are shown in Fig. 4.

The main class of the model `Arducopter` starts a `Scheduler` and a `Flight controller`. To improve model fidelity, sensor values are updated periodically by the scheduler to emulate the real update frequencies of the various sensors. The flight controller takes input from a pilot and from sensor data, on which sensor fusion has been performed, and is responsible for calculating desired accelerations for the UAV. These accelerations are translated, by the `Motors` class, into motor setpoints for each motor. The translation involves a complex tradeoff between roll, pitch, and yaw accelerations and total thrust, as well as motor clipping and voltage scaling. The `MotorsQuad` class is shown to illustrate that the `Motors` class can be extended to support any number or configuration of motors, thus increasing complexity even further.

[4] Note that sensor fusion is a very important aspect when considering robustness against noise and uncertainties in sensors.

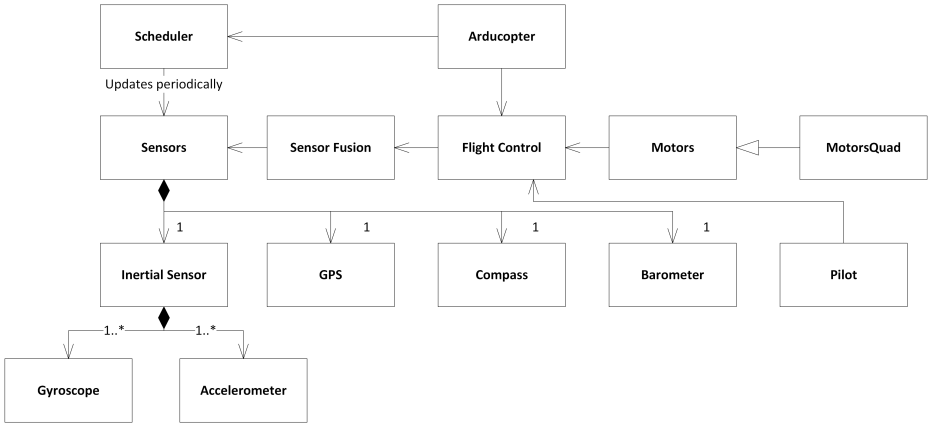

Fig. 3. Original DE model overview.

The `Flight Control` class contains the core functionality of the model and is probably also the most complex part (see Fig. 4 for an overview of the components). It can operate in multiple flight modes, which make use of either an attitude controller or both an attitude and a position controller. The attitude controller is capable of obtaining and maintaining any given attitude, whereas the modeled position controller is only capable of controlling the altitude and not the spatial location (latitude/longitude). In practice this means that it is not possible to make the UAV fly to a given location, but it is however possible to let the UAV move in any direction, simply in an uncontrolled manner. The attitude and position controllers depend on a number of low level controllers, such as Proportional-Integral-Derivative (PID) controllers, the simpler PI or P controllers, and a somewhat similar square root controller. To complicate things even further, most of these controllers are cascaded and added a feed-forward term in order to improve control capabilities. Additionally, a number of different filters are used to remove unwanted noise and vibrations caused by the fast spinning rotors.

5 Abstracting the Original Discrete Event Model

This section will investigate whether an abstraction to the original model described in Sect. 4 can be used for implementing vertical waypoint behaviour. The vertical waypoint behaviour was chosen, because the original DE model only supports vertical movement. In order to create a proper abstraction it is necessary to understand the task at hand. First, we consider the basic hardware of a UAV. This is shown in Fig. 5, which contains the following sensors and actuators:

Barometer: A barometer is used to measure atmospheric pressure and can be used to measure altitude.

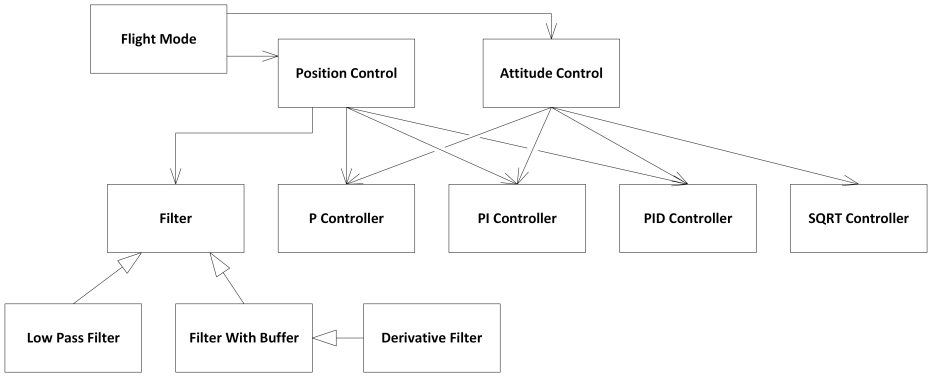

Fig. 4. Flight control components

Accelerometer: An accelerometer measures acceleration forces, and can be used to get the orientation of the UAV relative to earth's surface.
Gyroscope: A gyroscope measures angular velocity, e.g. yaw, pitch, and roll velocity and can therefore be used along with the accelerometer to get the orientation of the UAV.
Magnetometer: A magnetometer measures magnetic fields and can be used to detect earth's magnetic field, thereby providing a magnetic north.
GPS: A GPS can be used to measure altitude, latitude, and longitude and thereby provide a position in space.
Motor(s): The motor(s) are used for the rotor(s) on the UAV. One or more motors can be used.

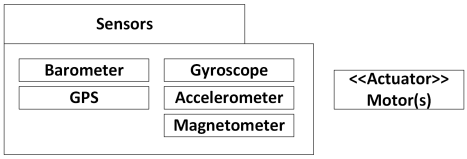

Fig. 5. Basic sensors and actuators of a UAV

Next, we consider the basic control flow of a UAV as shown in Fig. 6. This consists of the following components:

Sensor Input: Input from the sensors listed above.
Sensor Fusion: In order to gain useful information the sensor data must be fused together. For example, the accelerometer, gyroscope, and magnetometer is commonly referred to as an Inertial Measurement Unit (IMU). The information obtained from these three sensors is used to get the best possible position information, as they each have their strengths and weaknesses. The result of the sensor fusion is position and velocity.

Flight Mode: This construct is the control logic and it depends on the flight mode. A waypoint is a position in space, where the drone should fly to, and it can be followed by other waypoints. Pilot means that a pilot gives commands to the drone, e.g. throttle and yaw rotation. The output from the flight mode is roll, pitch, and yaw angles along with a desired velocity.

Motor(s): The throttle value is sent to one or more motors.

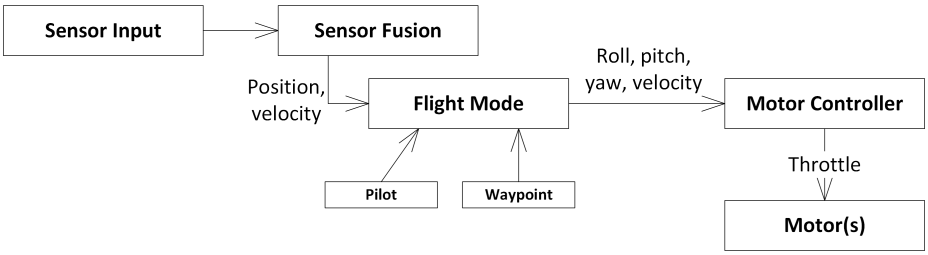

Fig. 6. Basic control flow of a UAV

The next step is to consider which parts to include in the abstraction in order to model the high-level waypoint behaviour for the UAV. High-level behaviour in the context of this paper is used to describe functionality closely related to the goal of a feature. For example, the goal of the waypoint behaviour is for the UAV to fly to a waypoint, and once this waypoint is reached it should fly to the next waypoint. Low level details are considered as the constructs that enables the high-level functionality, e.g. reading sensor data. The waypoint behaviour is based on the position of the drone, and once it coincides with the position of a given waypoint, the UAV has visited the waypoint. It only depends on the position and velocity of the UAV and not how they are retrieved. Thus the sensor fusion component can be abstracted away when developing high-level waypoint behaviour as GPS coordinates is enough to provide a position and calculate a velocity.

The abstraction introduced above is realised in an implementation that consists of two DE models that only differ in the feedback system. One of the models, henceforth referred to as the P model, uses a proportional gain controller. The other model, called the PID model, uses a PID controller instead of a P controller. Both models only make use of the GPS sensor component and the motor actuators. The P and PID controller is based on altitude and used to provide input to the throttle value. The model is shown in Fig. 7 and resembles the basic flow depicted in Fig. 6. The `UAV` class initiates and starts up the system. The GPS data is retrieved through the `GPS` class, and the `SF_GPS` class is created to provide an abstraction to the low level `GPS`, as sensor fusion is to be implemented later on. `PositionControl` contains the functionality to retrieve the GPS data, calculate a throttle value by converting the control value from `P/PID`, and to check whether the current position is within the bounds of

the current waypoint. The `FlightController` is responsible for the high-level functionality for waypoint behaviour. This means it uses `PositionControl` to calculate a throttle value and checking whether the current waypoint is within bounds along with setting a new waypoint once a waypoint has been reached. Finally, `MotorController` converts a single throttle value to a throttle value for each motor, represented by the class `Motor`.

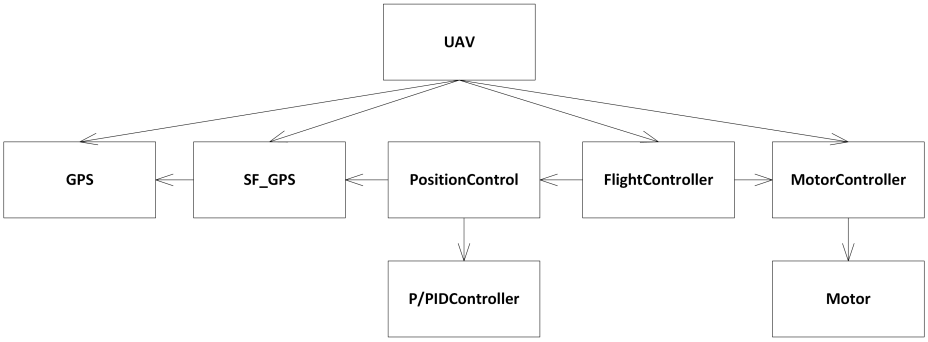

Fig. 7. The P and PID models.

A waypoint is considered reached with an approximation when the drone is vertically within 50 centimeters of the waypoint. The waypoints are located in an altitude of: three meters, two meters, and four meters. This combination forces the drone to respectively increase, decrease, and increase velocity. This is to ensure that the drone does not reach a subsequential waypoint because it overshoots the target. Overshooting means that the drone goes beyond its current target because of excessive velocity. The vertical waypoint behaviours of the three DE models is shown in Fig. 8. The figure shows that the drone reaches the waypoints in all three DE models with different levels of overshooting as shown in Table 1. The original model shows the best performance in terms of how fast it reaches the three waypoints. However, in general the PID model has the lowest total error with ~3.34 meters, where the P model has ~6.78 meters and the original model has ~3.89 meters. Because the P and PID model roughly resemble the original model they are useful for prototyping. Deciding whether a model is useful or not depends on the particular case. As co-simulation deals with approximations of real systems in order to achieve a reasonable simulation speed, it can be useful to define tolerances. A tolerance can be used to define whether a simulation result is "close enough" to the real system and therefore considered useful [15].

Another interesting measure is how "simple" these abstractions are compared to the original model. To compare the simplicity Lines Of Code (LOC) is used as measurement unit. The original model consists of 42 files with 2270 LOC along with an additional Java library, the P model consists of 12 files with 307 LOC,

and the PID model consists of 12 files with 333 LOC[5]. Thus the original model is roughly seven times larger than the P and PID model.

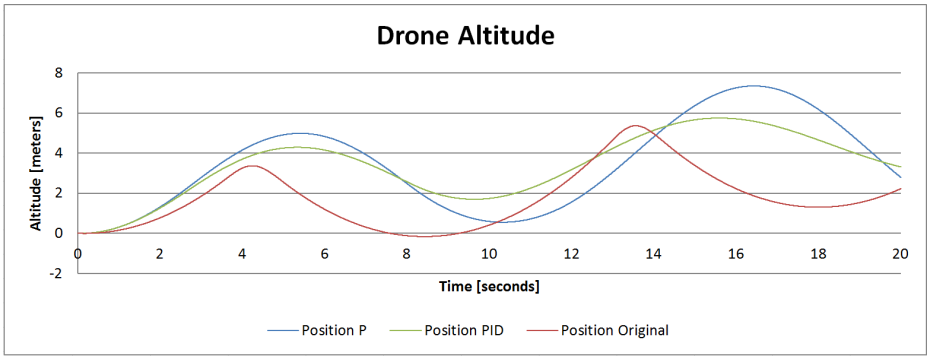

Fig. 8. The altitude of the drone following the waypoints. The waypoint functionality was implemented based on different abstractions.

Table 1. Overshooting for the three DE models.

Altitude Target	Model	Actual	Overshoot
3	P	4.983961	1.983960533
	PID	4.287995836	1.287995836
	Original	3.367945477	0.367945477
2	P	0.552304337	1.447695663
	PID	1.696441193	0.303558807
	Original	-0.161162787	2.161162787
4	P	7.345629313	3.345629313
	PID	5.745529086	1.745529086
	Original	5.363993294	1.363993294

The P and PID models were created by a software engineer within six hours without any prior knowledge of UAVs. This illustrates that using the model and prototype approach it is possible for a person without knowledge of mechanics to develop the high-level waypoint behaviour, and leave the development of low level details to experts within the given discipline. Implementing the waypoint behaviour in this fashion is an example of focusing on what should be accomplished rather than how. This leads to well defined interfaces, because the high-level behaviour, the "what", is implemented based on what the lower level details, the

[5] The original model does contain more functionality than the P and PID model, however the difference is significantly larger than what the additional functionality accounts for.

"how", should offer. It is illustrated in Fig. 9, where the interface to the sensor fusion component is already in place. The sensor fusion component can thus be refined without affecting the high-level waypoint behaviour[6]. The prototype models can also be used as a reference to the development of the low level functionality. In this case study it can be seen that the PID model has a smaller overshoot than the original model when moving from a high altitude to a lower altitude. Was this known before implementing the original model, then it might have affected the resources spent on tuning the variables of the used feedback systems.

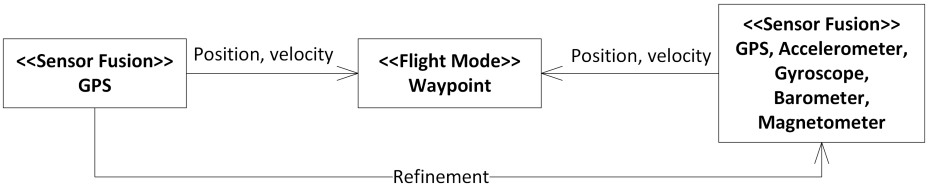

Fig. 9. Refinement of the `Sensor Fusion` component, without altering the interface to the high-level waypoint behavior.

It is important to consider the abstraction level when prototyping new features. Choosing the right abstraction level is an engineering skill that depends on the purpose of the abstraction. If the purpose is to prototype high-level behaviour, then choosing a too low abstraction level makes it a cumbersome task to prototype new features, as it is necessary to take low level details into consideration. However, if the purpose is to prototype low level details, then it is necessary to make these easily accessible and possibly abstract high-level code. Another important thing is not choosing a too high abstraction level if the goal is to code generate working code. In this case it can become difficult to unify the abstractions with the fully working implementation. The idea of separating abstractions into this sort of levels is described by Wing [16], who uses the term "layers". She states that we are working with at least two layers of abstraction simultaneously: the layer of interest and the layer below; or the layer of interest and the layer above.

Having a focus on abstractions and prototyping promotes a breadth approach rather than a depth approach as shown in Fig. 10. The breadth approach means that some can work on high-level behaviour using simple models of the low level details, while others work on the low level details. The depth approach means that part of the functionality is fully implemented before moving on. This requires low level details to be implemented, as they are the building blocks for the high level functionality. Therefore, the breadth approach can allow more people to work on different levels, however functionality might not be fully implemented as fast as when using the depth approach. Using the depth approach, some functionality

[6] It is not always the case that the same high-level component can be reused when other models have been refined.

will be fully implemented, whereas development of other functionality will not have begun. Thus, there are advantages and disadvantages to both approaches, and which approach to use depends on the given project. Another important aspect to consider is whether the effort spent on creating models and prototyping provide sufficient insight for the task to balance or exceed the resources spent [3].

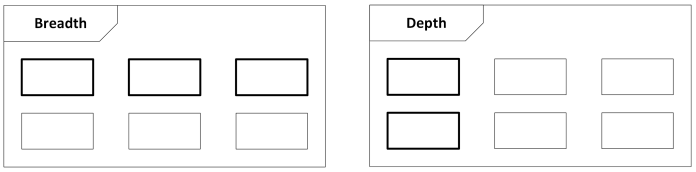

Fig. 10. Breadth and depth approach. The thick boxes represent fully implemented functionality, and the normal boxes represent shallow implementations, e.g. models or implementation to be done.

6 Transitioning to INTO-CPS Technology

The INTO-CPS toolchain offers the possibility of performing co-simulations with any number of FMUs. Because FMI is standardized the execution of co-simulations using INTO-CPS technology is not limited to models expressed in VDM-RT using Overture and 20-sim. Thereby any tool capable of generating an FMU can be used in the development process. This allows for choosing the tool best suited for a given task provided that the tool can generate an FMU. Furthermore, the INTO-CPS toolchain is being actively developed and supports verification, DSE, traceability, and other features. Therefore it is of interest to transition to INTO-CPS technology for the further development of the UAV, such that these features can be utilized. Furthermore, tools using FMUs can also be used in the development process e.g. MathWorks Simulink for low level control testing.

The transition from Crescendo to the INTO-CPS technology is aided by the tools 20-sim and Overture through support of FMU generation. The necessary modifications to the DE and CT model are limited to the contract or interface between the two models. When using the Crescendo tool, shared variables between the two models are declared explicitly in 20-sim as `imported` or `exported` `external` signals. In Overture such shared variables are declared implicitly as instance variables and linked in a separate link file. With the INTO-CPS tool chain and the use of FMUs, these interface declarations must be changed. In 20-sim the explicit declarations are no longer needed. To ease the transition between the two technologies, the model can be split into two connected 20-sim blocks; one with the interface declaration and one with the model functionality. By doing so, the model can still be used with the Crescendo technology and the

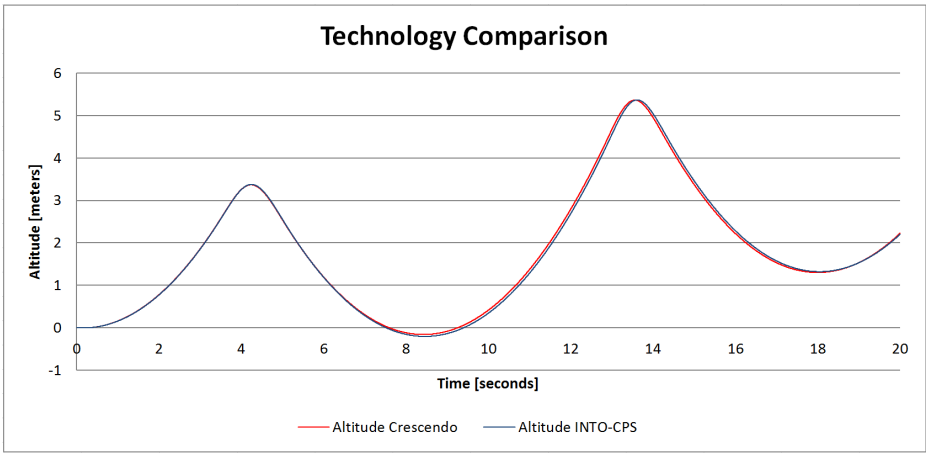

Fig. 11. Comparison of simulation results from Crescendo and the INTO-CPS tool chain

block containing the model functionality can be exported as an FMU and used with the INTO-CPS technology. In Overture shared variables must be declared using the construct `FMI Port`, which contains an instance variable. Therefore shared variables are now declared explicitely and they are accesable via get and set functions.

Once FMUs of both models have been created, they can be imported into the INTO-CPS app, where they can be connected, validated, and executed as a co-simulation. A comparison of the results from the original DE model run with Crescendo and the INTO-CPS app is shown in Figure 11. The comparison shows only minor differences between the two simulation technologies. This can be due to several reasons, e.g. varying step sizes in the two technologies. In the INTO-CPS app, the step size is determined entirely by the Overture FMU, whereas in Crescendo the 20-sim model generates additional intermediate steps, which seems to be related to the step size of the integrator in 20-sim. The data shown in Fig. 11 consist of ~8000 samples from the INTO-CPS app, and ~32000 samples from Crescendo. The FMU for the DE model is a tool wrapper for Overture and therefore the same code is executed but with an FMI layer on top, which can also make a difference. Additionally, it can also be due to the rounding performed by the co-simulation engine used in INTO-CPS.

7 Concluding Remarks

In this paper a full model of a DJI F450 Flameweel quadrotor UAV has been presented. This included the description of a CT model created with the tool 20-sim and a detailed DE model, called the original DE model, created with the open-source tool Overture. The CT model in 20-sim and DE model in Overture was coupled and co-simulated using the Crescendo tool. The CT model contains

all relevant electronics, mechanics, and sensors of the UAV. The original DE model is a reverse engineering of the APM:Copter project with irrelevant functionality abstracted away. It contains functionality to hold an attitude, hold an altitude, and respond to pilot instructions.

This paper also describes a case study, where the original DE model has been abstracted into two simplified DE models called the P model and the PID model. These models are simplified in the sense that they only use the GPS sensor and other low level details are abstracted away such as a low pass filter. The P and PID models only differ in their use of a P-controller and PID controller respectively. They both contain functionality to hold a given altitude and fly between vertical waypoints.

It has been shown that the abstracted models resemble the original DE model close enough for them to be used for prototyping. This was demonstrated by implementing vertical waypoint behaviour in all models and comparing their behaviour when flying to the three waypoints. Furthermore, the LOC required for all three implementations has been calculated to show the difference in implementation size. The results demonstrate that abstract models avoiding low level details can be used for prototyping purposes with reasonable results. This is interesting because it allows people with different expertises to work jointly on a project on different levels of abstraction, e.g. a software engineer working on high-level behaviour using models while a mechanical engineer works on low level constructs. The case study in this paper serves as an example of this, as it was implemented by a software engineer within six hours without particular knowledge of mechanics or electronics.

Finally the paper describes the effort of converting the CT and DE models to FMI compliant FMUs in order to be used for co-simulation in the INTO-CPS tool-chain along with a comparison of the results obtained with the two different technologies Crescendo and INTO-CPS.

8 Future work

The motivation of creating new DE models is to gain insight into developing control logic for a UAV. This insight will be made visible by documented models and data. The data will be gathered by utilizing DSE throughout the development process to various components and thereby choose optimal solutions. Part of this development will also result in building blocks in the shape of libraries for Overture, e.g. a vector library, and various generic components that can be used for other projects. An example of a generic component can be a feedback system such as a PID controller. During the development it will also be explored how to create the appropriate abstractions for testing and optimizing the constituent components. An example of creating the appropriate abstraction is related to a planned case study, which will investigate how to recover from battery failure. In this case low level details are important, whereas behaviour such as waypoint navigation is irrelevant.

The tools from the INTO-CPS project are expected to aid in the process of achieving optimal CT and DE models and ultimately to create a DE model detailed enough to be code generated and used to control a UAV. In this process alternative implementations will inevitably arise, and the choice of implementation will depend on results from experiments and case studies, e.g. using DSE. Thus it will contain functionality similar to the APM:Copter framework but with the difference that it will be documented. It is also the goal to enhance the CT model to a higher level of fidelity and thereby improve simulations.

Acknowledgments We would like to thank the anonymous referees for valuable input on this work. The work presented here is partially supported by the INTO-CPS project funded by the European Commission's Horizon 2020 programme under grant agreement number 664047.

References

1. Bouabdallah, S., Murrieri, P., Siegwart, R.: Design and control of an indoor micro quadrotor. In: Proceedings of the 2004 IEEE International Conference on Robotics & Automation. pp. 4393–4398. IEEE, New Orleans, LA (April 2004)
2. Broenink, J.F.: Modelling, Simulation and Analysis with 20-Sim. Journal A Special Issue CACSD 38(3), 22–25 (1997)
3. Fitzgerald, J.S., Larsen, P.G.: Balancing Insight and Effort: the Industrial Uptake of Formal Methods. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) Formal Methods and Hybrid Real-Time Systems, Essays in Honour of Dines Bjørner and Chaochen Zhou on the Occasion of Their 70th Birthdays. pp. 237–254. Springer, Lecture Notes in Computer Science, Volume 4700 (September 2007), iSBN 978-3-540-75220-2
4. Fitzgerald, J., Larsen, P.G.: Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edn. (2009), ISBN 0-521-62348-0
5. Fitzgerald, J., Larsen, P.G., Verhoef, M. (eds.): Collaborative Design for Embedded Systems – Co-modelling and Co-simulation. Springer (2014), http://link.springer.com/book/10.1007/978-3-642-54118-6
6. Grujic, I., Nilsson, R.: Model-based development and evaluation of control for complex multi-domain systems: Attitude control for a quadrotor uav. Tech. Rep. 23, Department of Engineering, Aarhus University (January 2016)
7. Isasa, J.A.E., Jørgensen, P.W., Larsen, P.G.: Hardware In the Loop for VDM-Real Time Modelling of Embedded Systems. In: MODELSWARD 2014, Second International Conference on Model-Driven Engineering and Software Development (January 2014)
8. Jensen, J., Chang, D., Lee, E.: A Model-Based Design Methodology for Cyber-Physical Systems. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International. pp. 1666–1671 (2011)
9. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-Integrated Development of Embedded Software. In: Proceedings of the IEEE. vol. 91, pp. 145–164 (2003)
10. Kramer, J.: Is Abstraction the Key to Computing? Communications of the ACM 50(4), 37–42 (2007)
11. Lausdahl, K., Coleman, J.W., Larsen, P.G.: Semantics of the VDM Real-Time Dialect. Tech. Rep. ECE-TR-13, Aarhus University (April 2013)

12. Lee, E.A.: Cyber Physical Systems: Design Challenges. Tech. Rep. UCB/EECS-2008-8, EECS Department, University of California, Berkeley (Jan 2008), http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html
13. Overture-Core-Team: Overture Web site. http://www.overturetool.org (2007)
14. Thompson, H. (ed.): Cyber-Physical Systems: Uplifting Europe's Innovation Capacity. European Commission Unit A3 - DG CONNECT (December 2013)
15. Thule, C.: Verifying the Co-Simulation Orchestration Engine for INTO-CPS. In: Doctoral Symposium FM 2016. Limassol, Cyprus (November 2016)
16. Wing, J.M.: Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 366(1881), 3717–3725 (2008)