Control and Scheduling Co-Design for a Simulated Quadcopter Robot: A Model-Driven Approach

Matteo Morelli and Marco Di Natale

Institute of Communication, Information and Perception (TeCiP)
Scuola Superiore Sant'Anna, Pisa, Italy 56124
{matteo.morelli,marco.dinatale}@sssup.it

Abstract. The Model-based development of robotics applications relies on the definition of models of the controls that abstract the computation and communication platform under the synchronous assumption. Computation, scheduling and communication delays can affect the performance of the controls in way that are possibly significant, and an early evaluation allows to select the best control compensation or the best HW/SW implementation platform. In this paper we show a case study of the application of the open T-Res framework, an environment for the co-simulation of controls and real-time scheduling, on a quadcopter model example, highlighting the possible tradeoffs in the selection of the scheduling strategy and priority assignment.

1 Introduction

Model-based development of robotics controls is an industrial reality. The MATLAB/Simulink tool from Mathworks is a very popular framework used to define the controls functionality and the model of the controlled plant, and provides for the simulation and verification of hybrid systems. In Simulink, however, the model execution is simulated according the the synchronous reactive paradigm, in which all the computations and communications are assumed to complete within the interval between two events in logical time (formally referred to as *synchronous assumption*). When the (controller) model is implemented in software and its implementation executes on a real architecture of CPUs and communication links, computation, scheduling and communication delays may exceed what is prescribed by the synchronous assumption and the jitters and latencies may affect the performance of the controls. The impact of these delays is often eveluated late, at testing time, with significant costs, additional development cycles and possible changes to the hardware architecture.

An early evaluation of the impact of the hardware and software implementation is desirable and requires the co-simulation of the controller functionality, the plant model, and the computation, scheduling and communication hardware and software platform, together with a model of the software tasks and the messages exchanged over the networks. To support such a co-simulation in the popular

Simulink environment we developed the T-Res open project and a framework supporting its use.

Our framework merges methods and tools of the MDE (Model Driven Engineering [15]) approach in a development flow in which Simulink models are used to define the functionality of the controls and SysML models define the hardware execution platform and the task model of the controls implementation. After the functionality is mapped for execution on the platform model, defining the structure of the tasks and messages, the execution and transmission times are estimated (or measured) and the Simulink model can be annotated with blocks that allow the simulation of the scheduling, computation and communication latencies, allowing to fine tune the control logic or the task and message model (possibly with their priorities), or evaluate different scheduling policies.

The evaluation of the impact of the scheduling on the controls performance allows to overcome the often myopic assumption that all control loops/tasks are of type hard real-time. In reality, several systems may miss deadlines without losing stability, and indeed, several systems (including fuel injection [8]) actually operate in spite of deadline misses, at the boundary of overload conditions.

MDE approaches have become popular in robotics and several MDE Integrated Development Environments (IDEs) and Domain-Specific Languages (DSLs) are available. BRIDE[1] is an IDE based on Eclipse developed in the BRICS project [1]. It targets the automatic generation of platform-specific code for component-based frameworks from a graphical (abstract) model of the system architecture and its SW components (the BRICS Component Model [6]). BRICS uses model-to-model (M2M) transformations to generate framework-specific code for the communication, configuration, composition and coordination of ROS [16] and Orocos-RTT [7] components. The declarative description of robotics architectures and software (SW) deployment using a DSL is described in [13] with a hierarchy of architectural concepts for hardware and software, inspired by AADL [4]. However, the properties of HW and SW that define the timing behavior of components are not included. The SmartSoftMDSD toolchain [17] supports non-functional properties for design-time real-time schedulability analysis. The framework allows the graphical modeling of applications in Papyrus[2], and provides M2M transformations to construct a platform-specific model for schedulability analysis using Cheddar [18].

Some IDE provide DSLs for the algorithmic description of behaviors. V^{3}CMM [5] is a modeling language that provides a simplified version of UML activity diagrams, to model the sequential flow of execution within components RobotML [11] is a DSL aiming at the design of robotic applications in Papyrus and their deployment to multiple target execution platforms (and simulators). It uses a specialization of UML state machines for the modeling of the behavior of generated component implementations. RobotML enables (simplistic) modeling of platform-specific non-functional properties of SW components, that are used to create models

[1] `http://www.best-of-robotics.org/bride/`
[2] `http://www.eclipse.org/papyrus/`

for third-party real-time schedulability analyzers. Hence, it suffers the same drawbacks of the SmartSoftMDSD toolchain.

Virtual Path [14] is a HW-SW co-Design method that includes Simulink in the development flow, to create executable models representing the controls. In [19], Wätzoldt *et al.* adapt the automotive toolchain to the development of robotic systems. The design methodology uses Simulink for the simulation of robot functionalities, and Embedded Coder for the generation of the implementation. AUTOSAR models and tools (e.g., SystemDesk [12]) are used to combine hard and soft real-time tasks in a system view and analyze the scheduling feasibility.

Finally, TrueTime [9] is a freeware Matlab/Simulink-based simulation tool that allows to model multi-task real-time kernels and networks in simulation models for networked embedded control systems and study the (simulated) impact of lateness and deadline misses on controls. Because of the monolithic architecture and the number of code artifacts that are needed for system configuration, the current TrueTime implementation is hardly compatible with an automatic model generation flow.

2 From the Simulink Model of the Controls to the Platform and Implementation Design

In the development flow considered in our work (summarized in Figure 1), a *Simulink functional model* of the controls executing in the abstract logical time (zero delays) is the starting point. The *functional model* is created by importing in EMF a Simulink model that includes the controller part and the model of the plant. The Simulink model must comply with the restriction that there is a decomposition level in which the controller part consists of a collection of subsystems and each subsystem only contains periodic blocks with the same period (each subsystem is single-rate).

A Matlab script uses the Simulink modeling API to parse the model structure and export an XML view of the controller subsystems. The XML conforms to a schema created in accordance with an Eclipse Ecore meta-model, defined for representing the execution constraints that apply to the Simulink subsystems and preserving the structural properties of the Simulink model, such as the types and interfaces of the blocks and the connections among the blocks, and also the information related to the timed execution events, including rate and partial order of execution constraints.

After the functional model is imported in the Eclipse EMF framework, a QVTo model-to-model transformation is executed to import the model in SysML, leveraging a profile definition for SR systems. Generic `Block` entities are mapped to standard SysML blocks; `Subsystem` entities are mapped to `SRSubsystem` instances. The input/output ports and the corresponding connections are suitably translated from the source model (Ecore) to the destination model (SysML).

Here, it is extended with the platform and mapping models. For the modeling of the hardware (HW) part of the *execution platform* we rely on the concepts provided in MARTE. It introduces the `HwProcessor` stereotype, which matches

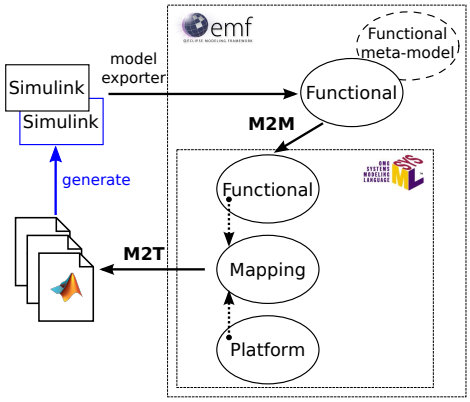


Fig. 1: The development flow for the proposed Model-Driven approach.

the concept of CPU and provides the attribute `nbCores` to specify the number of cores, thus enabling the modeling of multi-core architectures.

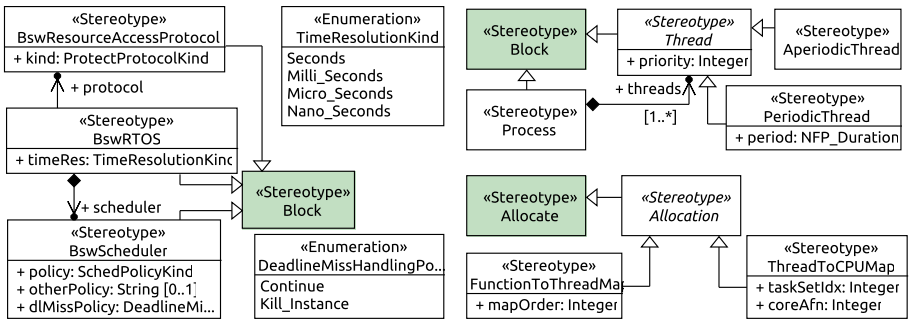
For the modeling of basic software (BSW) components and for the deployment of BSW modules onto the HW we define our own taxonomy of stereotypes, since those in MARTE are mostly cumbersome, come with a large number of properties and are in turn quite difficult to be mastered by the system designer. The `BswRTOS` package provides a set of stereotypes to model RTOS concepts (Figure 2a). The stereotype `BswRTOS` denotes an RTOS and inherits from the general block concept of SysML. The RTOS (kernel) contains a scheduler, denoted as `BswScheduler`, which is responsible for executing tasks according a given scheduling policy.

The *mapping model* represents the execution of functional elements by tasks and the allocation of tasks on cores. Concurrent execution contexts are classified in terms of `Process` and `Thread` instances. A `Thread` is contained in a `Process`, is characterized by a `priority` value and runs on one of the system cores under the control of an RTOS. Concrete specializations of `Thread` are `AperiodicThread` and `PeriodicThread` (with its `period`).

The mapping model also specifies a set of dependencies that define mappings/deployments as extensions of the standard SysML Allocation concept. The `FunctionToThreadMap` denotes the mapping of a functional subsystem into a `Thread`. When multiple subsystems are mapped into the same `Thread`, the attribute `mapOrder` defines how their execution will be serialized in the generated thread code. The SysML profile for mapping is shown in Figure 2b.

3 Time and Resource Aware Simulation in Simulink

The simulation of the control functions considering task implementations with finite execution times and RTOS scheduling delays is enabled by integrating Simulink and a RT scheduling simulator in the T-Res [3] co-simulation framework. T-Res is designed according to object-oriented design patterns to provide an easy integration with any RT simulator.



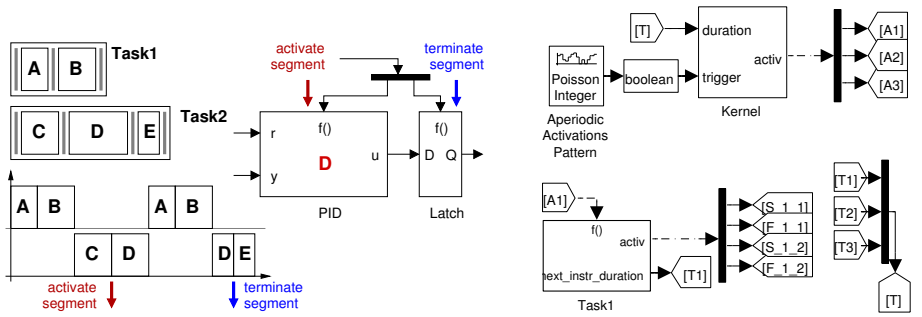
(a) Structure of the `BswRTOS` meta-model. (b) Structure of the `Mapping` meta-model.

Fig. 2: `Mapping` and `BswRTOS` meta-models.

The Simulink master simulation engine computes the model updates at *major steps*: time instants in which the inputs and outputs of the blocks are updated. Major steps include all the periodic activation times of tasks, as well as the aperiodic events that lead to the activation of other tasks.

T-Res is implemented as a set of custom blocks that execute at all major steps, interact with the Simulink engine and capture all the relevant events from the simulated environment. Every time a major step occurs, the block implementing the RTOS kernel is invoked and processes (if there is any) the task arrival events. These events are forwarded to the underlying RT scheduling simulator and cause an update of its internal structure. Then, the kernel block queries the scheduling simulator to determine future events (execution completions and context switches) and uses the Simulink API to define major steps in the simulation at all the points in time in which a task scheduling event occurs.

In a RT simulator, tasks execute according to a model of (time-consuming) computations. T-Res assumes that the execution of a task is split in units that are atomic from the standpoint of execution time granularity, but can be preempted, called *segments*, informally corresponding to the execution of a function called by the task main code. Each segment is identified by an execution time and all segments in a task are executed according to a pre-defined sequence. Segments represent the execution of Simulink subsystems and their execution order in a task must match the order of execution of subsystems. The time duration of each segment corresponds to the execution time of the code implementing the subsystem. The start and completion times of the segments correspond to the times in which the corresponding subsystems read or sample their inputs and produce their outputs. The activation of the Simulink subsystems is changed from periodic to function activated and a latch barrier is added on all their outputs. Figure 3a shows the activation mechanism: when a segment starts executing, the subsystem is activated; the output signals are latched and enabled when the segment terminates. The signals activating a subsystem (and its input sampling) and its output latch are generated by the task blocks upon the beginning of the execution and the completion of the task segment.



(a) The execution of subsystems modeled through segments.

(b) `Kernel` driving three `Task` blocks (top), and `Task` with two segments (bottom).

Fig. 3: Execution model of segments and interfaces of T-Res blocks.

The actual implementation of T-Res relies on two custom blocks, namely `Kernel` and `Task`, implemented as C++ S-Functions. Blocks' input/output interfaces are shown in Figure 3b. The block `Task` models one instance of a task that consists of the serialized execution of the segments/subsystems. `Task` is a triggered subsystem, executed on the occurrence of a function call event received on its port `f()`. Its output port `activ` issues activation and termination events to the task segments; port `next_instr_duration` outputs a scalar signal representing the duration of next segment executed by the task. The duration of segments is set by a variable in the Matlab workspace. The computation time of a segment can be fixed or random (e.g., uniform and exponential distributions).

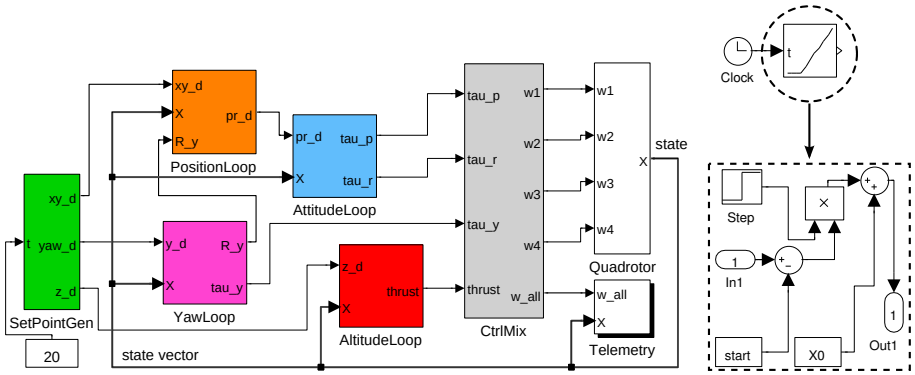
The block `Kernel` models an event-based RT kernel and the scheduler inside it on a single- or multi-core computer node. It is responsible for keeping the scheduling simulation aligned with the system simulation. At each activation, it checks for any aperiodic requests. If there is any, it activates the corresponding aperiodic tasks. Next, it advances the RT scheduler simulator. Two types of events are relevant for the simulation: the *segment completion* and *task completion*. In the first case, `Kernel` reads the input signal on the port `duration` and dynamically creates a new instruction for the corresponding task. In the second case, `Kernel` resets the internal state of the corresponding task. A number of parameters configure the (simulated) kernel such as the scheduling policy and the number of cores of the computer node. Parameters are set through the `Kernel` mask dialog. T-Res is open-source and is released under the terms of the 3-Clause BSD License. Currently, it features a concrete implementation of the adapter layer based on RTSim [2], which is available under the GNU GPLv2+.

4 Case Study: Quadcopter Attitude Control

The application of the methodology and an example of analysis results are shown using a model of a quadcopter.

4.1 Functional, Platform and Mapping Models

The quadcopter is required to lift off and fly in a circle at constant altitude, while spinning slowly around its Z-axis. The adopted control scheme (shown in Figure 4a) is taken from [10] with minor changes introduced to comply with our design restrictions. The original model in [10] contains multiple functional loops at the top level of the model hierarchy dedicated to set-point generation and flight control. Each loop has been included in a Simulink subsystem. The constantly increasing signal for the desired yaw angle, originally generated by a `Ramp` block in [10], is now obtained from the set of blocks of Figure 4b that use the output of an external `Clock` block as time source. In Figure 4b, `start` represents the time at which the block begins generating the signal, `X0` is the initial value of the output and the the rate of change of the generated signal is influenced by the parameters of the block `Step`. This is because subsystems mapped into segments cannot contain continuous time blocks (such as `Ramp`).



(a) Control scheme from [10], organized in subsystems.

(b) Ramp using external time-source.

Fig. 4: Models used for the quadcopter flight-control scheme.

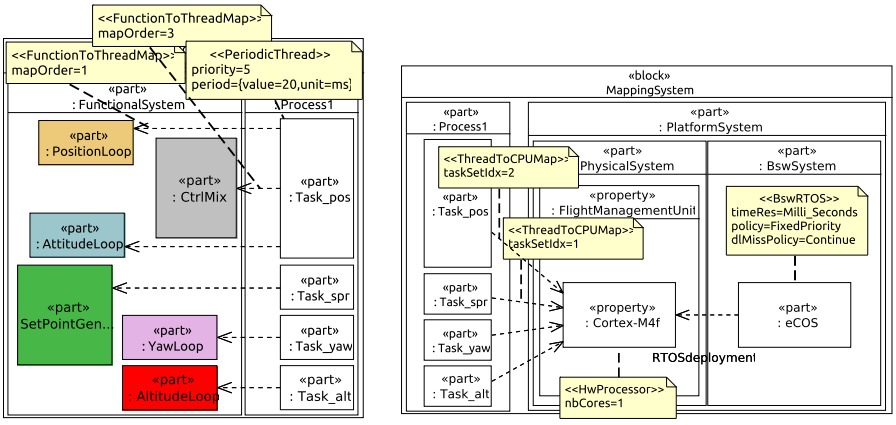
The set-points of the desired circular path and the desired yaw and altitude are generated by the subsystem `SetPointGen`. `Quadrotor` implements the motion of vehicle. The inputs are the speeds of the four rotors; the output is the 12-element state vector with the position, velocity, orientation and orientation rate of the quadcopter. The actual vehicle velocity is assumed to be estimated by an inertial navigation system or GPS receiver (i.e., there is no velocity estimator in the Simulink model).

The control strategy involves multiple nested loops that compute the required thrust and torques so that the quadcopter moves to set-points. Position control has a two-level hierarchical structure: the subsystem `AttitudeLoop` implements the inner loop, which uses the current and desired roll and pitch angles and angular rates to control the vehicle's attitude and to provide damping (to slow down the dynamics). The subsystem `PositionLoop` realizes the outer loop, which controls

the XY-position of the flyer by generating changes in roll and pitch angles so as to provide a component of thrust in the direction of the desired motion. Finally, yaw angle and altitude are controlled by proportional-derivative (PD) controllers, respectively implemented by the subsystems `YawLoop` and `AltitudeLoop`.

In practice, control loops are implemented as real time tasks, with finite execution times, running at different rates under the control of a scheduler. Typical execution rates range from 10Hz for reading (generating) set-points to 50Hz (or more) for controlling the vehicle attitude. To investigate how the performances of control code are actually affected by computation and scheduling delays, a structural view of the Simulink model of controls is first exported to Ecore and then *automatically* translated into a SysML model in Papyrus, where it is extended with the models of platform and mapping (deployment).

Figure 5a shows a (partial) view of an implementation model of controls consisting of four periodic tasks. `Task_spr` runs every 100ms and reads the set-points. `Task_pos` uses the set-points and the current state of the vehicle to perform the position control. Every 20ms, it executes the position loop, the attitude loop and the control mixer, in sequence. Finally, `Task_yaw` and `Task_alt` use the same information to perform yaw and altitude control with a period of 50ms and 25ms, respectively. Figure 5b shows a view of the deployment model of tasks to a single-core Autopilot/FMU board running a Fixed Priority (FP) real-time scheduler. Subsystems are now modeled as executing with execution times randomly generated according to uniform distributions (Figure 7b). Once the task priorities are specified, the mapping model includes all the information needed to automatically generate and connect the kernel and task blocks to the original Simulink model.



(a) Function-to-task mapping model. (b) Models of Autopilot board (with BSW) and of task-to-platform mapping.

Fig. 5: Functional, platform and mapping models.

4.2 Back-Annotated Functional Model

The Acceleo M2T transformation processes the mapping model and generates a collection of Matlab scripts that contain the back-annotation commands. The execution of the Matlab scripts produces the Simulink model of Figure 6.

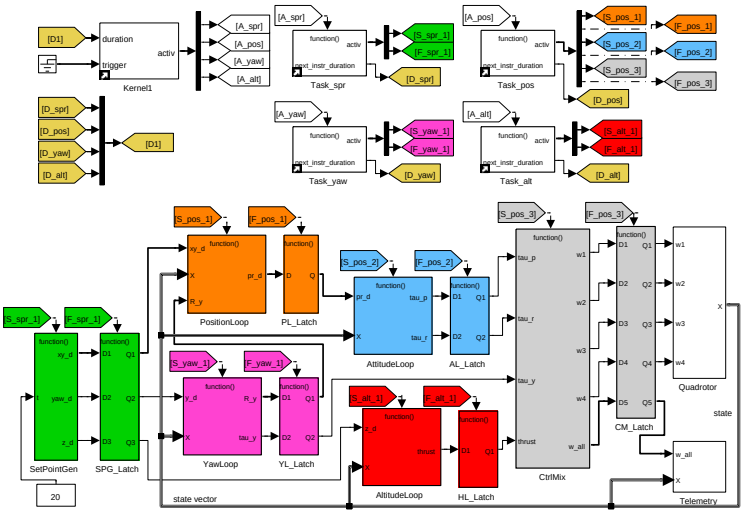


Fig. 6: Attitude control with models of RT kernel and tasks from T-Res.

One instance of kernel block (`Kernel1`) and four instances of task blocks (the names are the same of the corresponding SysML modeling artifacts) are added to the functional model. `Kernel1` outputs task-activation signals in the order specified by the `taskSetIdx` attributes of the `«ThreadToCPUMap»` allocations in the mapping model. Activation signals are forwarded to task blocks through the `Goto-From` connections labeled `A_spr`, `A_pos`, `A_yaw`, and `A_alt`. Similarly, tasks communicate the duration of the next segment to `Kernel1` through the `Goto-From` connections labeled `D_spr`, `D_pos`, `D_yaw`, and `D_alt`.

Each control subsystem is transformed to a triggered subsystem and a latch barrier is added on its outputs. Task blocks manage the activation and termination signals of the (control) subsystems executing in the segments.

Figure 7a shows a snapshot of the generated Matlab code that adds the `Kernel1` block, and configures its parameters. All parameters are available from the platform model and their values are set through the Matlab function `set_param()`. The timing properties and the type of tasks in the task-set are described by the variable `task_set_descr` in Figure 7b (cell array). Tasks types and periods (or interarrival times) are available from the mapping model. Relative deadlines coincide with periods and activation offsets are set to zero. All task periods are expressed in *milliseconds*, therefore a time-scale constant equal to 0.001 is generated. The duration of segments executed by each task is described by a Matlab cell array of strings (Figure 7b). Each string that describes

```
% - Add and configure the Kernel block
kern1 = 'quadcopter_bn/Kernel1';
add_block('t_res/Kernel', kern1);
set_param(kern1, 'taskset_descr_name','task_set_descr');
set_param(kern1, 'scheduling_policy', 'FIXED_PRIORITY');
```

(a) Matlab commands for the generation of block `Kernel1`.

```
% Description of timing properties of task set
task_set_descr = {...
  % type          %iat       %rdl       %ph  % prio
 'PeriodicTask', 100*0.001, 100*0.001, 0.0, 0; ...  % spr
 'PeriodicTask', 20*0.001,  20*0.001,  0.0, 5; ...  % pos
 'PeriodicTask', 50*0.001,  50*0.001,  0.0, 15; ... % yaw
 'PeriodicTask', 25*0.001,  25*0.001,  0.0, 10; ... }; %alt

% Sequences of pseudo instructions (task codes)
spr_instrs = {'delay(unif(0.001,0.002))'};
pos_instrs = {...
 'delay(unif(0.005,0.008))'; ... % PositionLoop
 'delay(unif(0.003,0.007))'; ... % AttitudeLoop
 'delay(unif(0.002,0.004))'; ... };% CtrlMix
yaw_instrs = {'delay(unif(0.004,0.006))'};
alt_instrs = {'delay(unif(0.008,0.009))'};
```

(b) Definition of type and timing properties of tasks.

Fig. 7: Matlab commands for the configuration of kernel and task blocks.

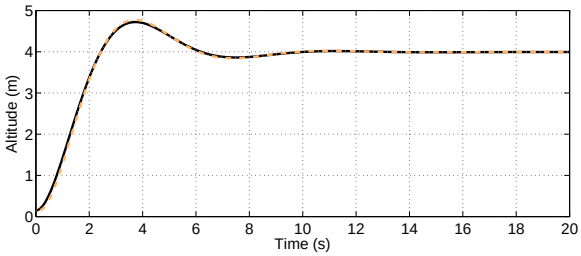
the computation time of a segment is available from the `execTime` attributes of the `«SRSubsystem»` block instances.

4.3 Scheduling Selection and Priority Assignments

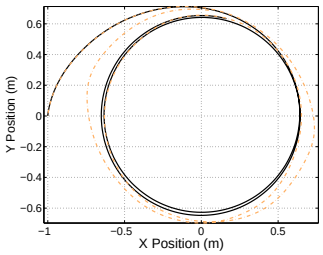
All design refinements, be them minor (e.g., changing the scheduling policy) or more prominent (e.g., mapping functional subsystems to a different task-set), are realized at SysML level to keep platform and mapping models in synch with the generated Matlab code for back-annotations.

Initially, `Task_spr` is given the highest priority; the other tasks' priorities are assigned according to their period, so that the shorter the period the higher the priority (Rate Monotonic rule). In this case, computation times and scheduling delays induce deadline misses of tasks `Task_yaw` and `Task_alt`, that do not affect much the altitude control, as shown in Figure 8a, but degrade the performances of circular path-following significantly (Figure 8b). This fact is easily explained if one considers that the low-priority task `Task_yaw`, which drives the high-priority task `Task_pos` (that controls the XY-position of the flyer), is repeatedly subject to preemption from the mid-priority task `Task_alt`, and that this prevents the preservation of SR communication flows between `Task_yaw` and `Task_pos`, with respect to the pure functional control model of Figure 4a.

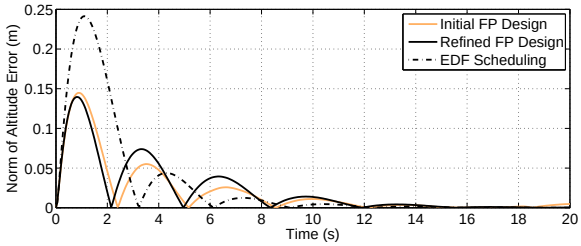
The analysis indicates that the response time of task `Task_yaw` has a significant impact on the effectiveness of the control action, and suggests to raise its priority to a value greater than the one of `Task_alt`. Figure 8d shows the simulation results of circular path-following in the refined design. `Task_yaw` has now a



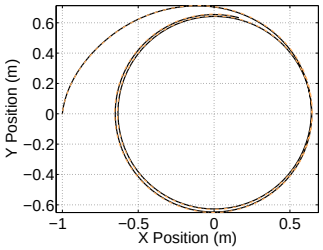
(a) Altitude control.



(b) Circular path-following of initial design.



(c) Norm of altitude error with respect to the pure functional control.



(d) Circular path-following of refined design.

Fig. 8: Simulation results. Comparison of trajectories and errors for different scheduling and priority assignments.

priority level greater than `Task_alt` and meets all its deadlines; consequently, the control behavior is closer to the pure functional one. On the other hand, `Task_alt` misses more deadlines than in the initial design and the altitude control performs slightly worse, as shown in Figure 8c (dark line vs light line). However, it is still controlled with a reasonable error, which makes the refined design preferable. As an additional option, we tried an application of the Earliest Deadline First (EDF) dynamic scheduling policy, which results in a slightly worse performance of the altitude control (dashed line of Figure 8c) and path following performance similar to that of the refined priority model (not shown in the graphs but practically overlapping with the dark line).

5 Conclusions And Future Work

The paper presents a case study application of a MDE framework for the definition of the execution platform and the impact of the computation and communication dlays and the T-Res co-simulation framework to a quadcopter model. The example shows how the selection of task priorities and scheduler models can affect the performance of the controls and the co-simulation environment allows to quantify the errors for different options. Future work includes the extension of the modeling and co-simulation framework to networked architectures and messages and the evaluation on a distributed case-study.

References

[1] BRICS - Best practice in robotics, http://www.best-of-robotics.org/
[2] RT-Sim – Real-Time system SIMulator, http://rtsim.sssup.it/
[3] T-Res – Time and Resource Simulator, http://retis.sssup.it/tres/
[4] Architecture Analysis & Design Language (AADL), http://standards.sae.org/as5506b/
[5] Alonso, D., Vicente-Chicote, C., Ortiz, F., Pastor, J., Alvarez, B.: V3cmm: a 3-view component meta-model for model-driven robotic software development. Journal of Software Engineering for Robotics 1(1), 3–17 (2010)
[6] Bruyninckx, H., Klotzbücher, M., Hochgeschwender, N., Kraetzschmar, G., Gherardi, L., Brugali, D.: The BRICS Component Model: A Model-based Development Paradigm for Complex Robotics Software Systems. In: Proc. of the 28th Annual ACM Symposium on Applied Computing. pp. 1758–1764 (2013)
[7] Bruyninckx, H., Soetens, P., Koninckx, B.: The Real-Time Motion Control Core of the Orocos Project. In: IEEE International Conference on Robotics and Automation. pp. 2766–2771 (2003)
[8] Buttle, D.: Real-time in the prime-time. Keynote presentation, Euromicro ECRTS Conference 2012 (July 2012)
[9] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Årzén, K.E.: How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime 23(3), 16–30 (June 2003)
[10] Corke, P.I.: Robotics, Vision & Control: Fundamental Algorithms in Matlab. Springer (2011)
[11] Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M.: Robotml, a domain-specific language to design, simulate and deploy robotic applications. In: Proc. of the 3rd Intl. conference on Simulation, Modeling, and Programming for Autonomous Robots. pp. 149–160 (2012)
[12] dSPACE GmbH: SystemDesk, \url{http://www.dspace.com/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm}
[13] Hochgeschwender, N., Gherardi, L., Shakhirmardanov, A., Kraetzschmar, G., Brugali, D., Bruyninckx, H.: A Model-Based Approach to Software Deployment in Robotics. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. pp. 3907–3914 (Nov 2013)
[14] Nickl, M., Jörg, S., Hirzinger, G.: The virtual path: The domain model for the design of the MIRO surgical robotic system. In: 9th International IFAC Symposium on Robot Control, IFAC. Gifu, Japan. pp. 97–103 (2009)
[15] Object Management Group: Model Driven Architecture (MDA), http://www.omg.org/mda/specs.htm
[16] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T.B., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
[17] Schlegel, C., Steck, A., Brugali, D., Knoll, A.: Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering. In: SIMPAR. pp. 324–335. LNCS, Springer (2010)
[18] Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. In: Proceedings of the ACM International Conference on Ada (SIGAda). pp. 1–8 (2004)
[19] Wätzoldt, S., Neumann, S., Benke, F., Giese, H.: Integrated Software Development for Embedded Robotic Systems. In: Proc. of the 3rd Int. Conference on Simulation, Modeling, and Programming for Autonomous Robots. pp. 335–348 (2012)