# A Robotic Vehicle Testbench for the Application of MBD-MDE Development Technologies

M. Morelli[1] Federico Moro[2] Tizar Rizano[2] Daniele Fontanelli[2] Luigi Palopoli[2] M. Di Natale[1]

***Abstract*— Models are used in control domains for early validation of system properties, using simulation or formal verification, and for the automatic generation of a software implementation. We propose an approach in which a functional model of the controls is matched to a model of the execution platform through an intermediate mapping model, that represents the software tasks and communication messages. The functional model is (partly) developed in Simulink and code is generated for each subsystem. Next, an abstract view of the functional model is imported in SysML. Using SysML, a model of the execution platform is created, and an implementation of the subsystems as a set of tasks and messages is defined and evaluated. The M2T Acceleo tool processes the mapping model and generates the Orocos-compliant task code executing the C/C++ functions generated from Simulink, and the inter-task communication. This paper outlines the proposed flow and provides the description of a robotic car testbench used to show the application of the methodology. The testbench has enough functional complexity and a distributed implementation to justify the creation of architecture models, while requiring a moderate cost and effort for its construction by the interested researchers.**

## I. Introduction

Models are used in control domains such as automotive and avionics for early validation of system properties, using simulation or formal verification, and for the automatic generation of control software. Model-Based Design (MBD) tools such as Simulink [1], based on a synchronous-reactive (SR) execution semantics, allow the modeling and simulation of hybrid systems, in which functionality is represented using an extended finite-state machine formalism. Early verification of the system functionality is also valuable in many robotics systems, which perform complex actions demanding a timely, predictable and certifiably safe behaviour.

MBD modeling tools represent the control functionality in abstract terms, according to a set of events in logical time. When the functionality is implemented on a distributed computing platform that executes the controls as a set of software tasks and network messages, the *synchronous assumption* (conformance with respect to the model execution semantics) should be preserved against computation and communication delays [2]. Alternatively, the development process should include a suitable model of the execution platform and the controls implementation to evaluate the delays and estimate their impact on the controls performance (or the feasibility against deadlines). This requires a detailed model of the control implementations and the execution hardware and software [3], for which languages and tools are not readily (commercially) available.

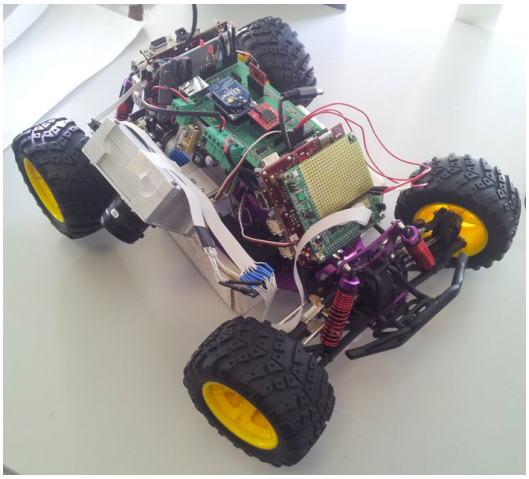

Fig. 1: The robotics car testbench.

We believe that MDE tools and methodologies (including the Eclipse Modeling Framework EMF and SysML) are suited for the specification of the execution platform and the software and message implementations. A development flow that integrates MBD and MDE technologies can provide the appropriate set of models for the automatic generation of implementations and the analysis of functional and time properties.

As a testbench for the methodology we propose a car-like robotics application. The system has sufficient complexity to justify the development of significant functionality using a MBD flow, including image processing, supervisory controls and low-level control loops. The execution architecture is distributed and leverages computation boards and communication technologies that are widely available. Because of its complexity and the distributed execution platform, the software and messaging architecture is not trivial and justifies timing analysis (schedulability and communication) providing an additional dimension to the problem.

Overall, the testbench (a picture is in Figure 1) provides an inexpesive platform to validate controls, development and analysis methodologies and integration of heterogeneous models.

In this WiP paper, we first outline our methodology,

[1]TECIP Institute, Scuola Superiore S. Anna, Pisa, Italy, [2]DISI, Universitá degli Studi di Trento, Italy

then the testbench and how the methodology can be applied to its development. To facilitate the dissemination of information, models and code of the open testbench, a web site will be created as a platform for exchanging artifacts, support discussion forums and share results.

## II. Integration of MBD and MDE in the Development Flow

In our proposed approach, a functional model of the system controls is matched to a purposely constructed model of the execution platform through an intermediate mapping model, that represents the software tasks and messages (local or on the network) that realize the functions.

A natural candidate for the functional modeling is the Simulink/Stateflow synchronous language. However, functionality could also be implemented directly as code. In both cases, an abstract view of the functional model is required. This abstract view should account for all the information related to the (timed) events that are relevant for the system, including rate constraints, partial order of execution constraints and any other synchronization constraints.

For the model of the execution platform, we are going to use the open source SysML Papyrus tool, which operates on top of the Eclipse Modeling Framework. On top of the modeling features offered by the standard SysML metamodel, we will provide a domain-specific profile for robotics applications that leverages (and extends) the standard OMG MARTE profile [4] for embedded and real-time systems. In addition, our profile will represent the common execution hardware in use in robotics systems and, on top of it, the Orocos-RTT framework [5], which encapsulates communication and RTOS services and acts as a middleware.

In addition to a dedicated profile for the model of the execution architecture, another profile is created to represent the model of the software implementation. This model consists of the set of all tasks and messages implementing the system functions and the communication signals. When a functional model is put in correspondence with an execution architecture, the task and message model is produced as the result (either by hand by the designer or as the result of the operation of synthesis tools). The profile extensions allow to define the mapping and evaluate the computation and communication delays, but are also used for the automatic generation of a concurrent task implementation on top of Orocos. Code generation is performed using the standard (and open source) Model-to-Code transformation tool Acceleo.

These are the process steps. A functional model of the controls is developed in Simulink and code is generated for each subsystem. Next, an abstract view of the functional model is generated and imported in the Ecore framework and then in SysML using a model-to-model (QVT) transformation. In the SysML modeling environment, a model of the execution platform is created, and an implementation of the subsystems as a set of tasks, executing on the platform nodes and exchanging messages is defined.

The task model may be constrained in such a way that only flow-preserving implementation of the functional model are allowed. Next, the Acceleo tool processes the mapping model and generates the Orocos description of the tasks and the inter-task communication according to the specification. The Orocos tasks execute the C/C++ functions, generated from Simulink and implementing the control subsystems.

The resulting methodology is a merger of the MBD and MDA paradigms. The overall tool flow is represented in Figure 2.

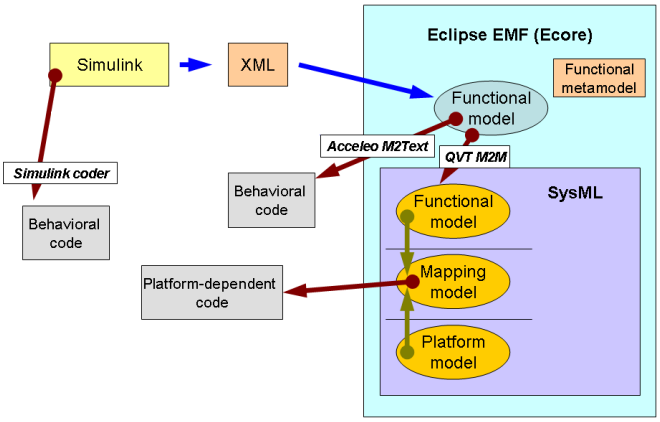


Fig. 2: Exchange of information and code generation by the framework tools.

## III. Car-Like Robotic Testbench

The robotic car testbench provides a case study for reasoning about all the system levels: functional, execution platform and software implementation. In the following, we first describe the functional model by highlighting the main functional components/subsystems. Then, we will provide a general description of the execution platform and the software architecture.

**Functional View.**

The functionality developed for the car system consists of a path follower, to be possibly further extended with more complex features. The car has four weels and two cameras (front and side) and moves in a physical environment consisting of a road, external objects and agents. Sensors collect information (in our case position, attitude and velocity of the car), and actuators control the engine and the steering wheels. Sensors and actuators are involved in low-level feedback control loops targeting setpoints on a given trajectory. A *planner* decides the line to follow, which is implemented as a sequence of manoeuvres such as go straight with speed X, turn with radius Y and speed Z etc. Planning decisions are made based on performance metrics, such as saving time, saving energy, overtaking slower vehicles, avoiding fixed

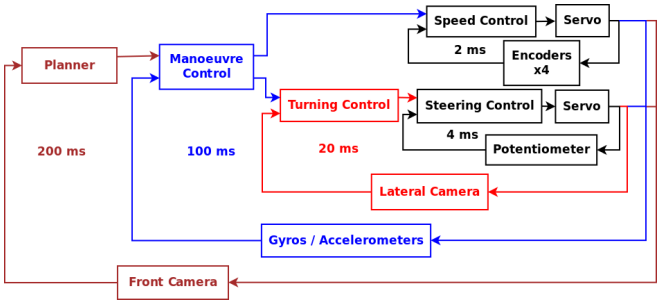


Fig. 3: Sensing and actuating control loops in a nested structure.

obstacles etc. The decided path is tracked by the low level controllers.

From a high level perspective, the system consists of a set of nested control loops, as in Figure 3, with the planner being the driving block of the entire system. Each control loop is activated periodically and has a different frequency/period, as shown in Fig. 3.

At the low level the robotic car has two servos controlled by a PWM signal: one is the engine that moves the car and one controls the steering angle. Each wheel has a relative encoder for speed monitoring, and a potentiometer is mounted in front of the car to provide feedback on the steering position. The sensors are used by two low level feedback loops (with a period of $2ms$ and $4ms$) implementing the PID controllers on the speed and the turning angle. A basic Inertial Platform, composed by gyros and accelerometers, completes the set of low level sensors and is used to improve the estimate of the car position.

A line following algorithm controls the position of the car with respect to the ideal panned trajectory. The algorithm used data from a high frame rate camera, pointing sideways, which estimates the position and attitude of the car with respect to the road line. The line following algorithm and the speed controller receive the set points from a manoeuvre controller that defines the sequence of manoeuvres and monitors their execution.

A second camera, mounted on the front of the vehicle, is used for path reconstruction and obstacle detection. This camera is activated with a relatively low rate (5 frame per seconds). The Planner receives an image from the camera, reconstructs the path and extracts other meaningful information (e.g., on the presence of obstacles). Then, it decides the vehicle manoeuvre and communicates it to the Manoeuvre Controller.

The vision algorithms used for each camera are a combination of Randomised algorithms (RANSAC) and Kalman Filtering [6], [7] generating a widely changing computing workload, difficult to manage with the standard tools of digital control [8].

In addition to highlighting essential mathematical and physical aspects related to the stability of the system and to the correctness of the design, the models composing

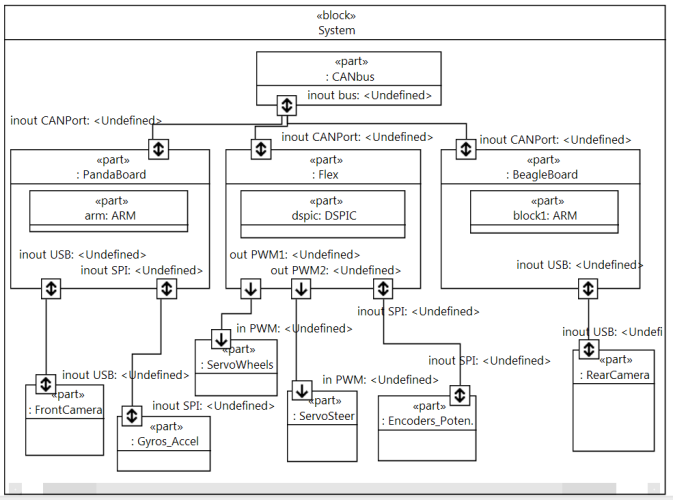


Fig. 4: The hardware architecture of the system.

the functional diagram provide a possible decomposition into subsystems, a definition of their relations and of the timing constraints for their execution.

**Hardware Architecture.**

The computing system consists of three boards connected by communication buses. The first board is a FLEX development board based on a 16 bit dsPIC. The board is provided with an OSEK-compliant RTOS (Erika). The PIC microcontroller can be connected to external devices by means of common digital interfaces like low power communication systems (SPI or $I^2C$), or PWM. The FLEX also supports advanced communication technologies like Ethernet and CAN bus.

The other two components are two ARM evaluation boards: a Beagleboard and a Pandaboard. Both are Texas Instruments products based respectively on OMAP3 and OMAP4 processors version. The RTOS used for these components is a Linux kernel modified with RTpreempt patches, to improve its real–time performance. The boards have SPI and $I^2C$ interfaces for connection with sensors and other low level peripheral; other connectivity solutions are the classic USB, Ethernet, Bluetooth and IEEE802.11 which facilitate remote control and telemetry. The Beagleboard and Pandaboard provide enough computing power to support the camera processing functionality, but have a limited power consumption and a low cost, both desirable features for robots used in laboratory activities.

Fig. 4 shows a view of the overall execution architecture using an example (and early) version of SysML modeling.

The processing units are connected through a CAN BUS, which offers a sufficient bit-rate for our applications without incurring the cost of an Ethernet switch in terms of power consumption. Other communication technologies are used for sensors and actuators interface. In general, the FLEX board is adopted for basic functionalities of the system: motor and steering controllers, and therefore for the communication with low level sensors and actu-

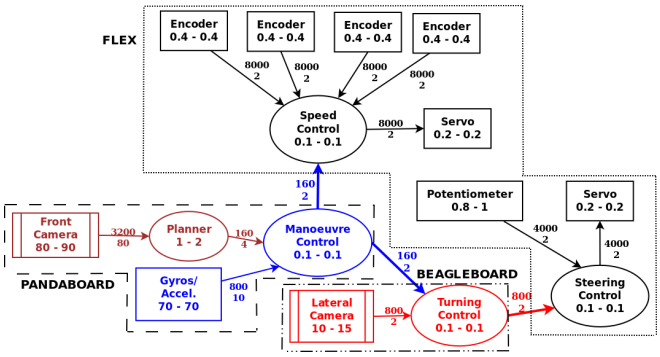


Fig. 5: The task set generated from the functional diagram. Each task is labeled with its average and worst case execution times in ms. Edges represent messages with payloads (bytes). The diagram also shows the allocation of the tasks on the computing units.

ators. Gyros and accelerometers are connected with the Pandaboard.

The two cameras, seen as high level devices, have an USB connection to the two ARM boards. Usually, ARM processors design integrates in the system some co-processing units, like GPUs or DSPs, which extend the capabilities of this kind of processing units. In particular, it is possible to increase the performance of algorithms, such as the ones used in our study case. This reinforces the choice of using microprocessors to define a level of sensors and actuators interaction, and exploit more advanced processors for the high level intelligence of the system. A common bus between the processing units facilitates the data flow for all the tasks running in the system.

**Software Architecture and Mapping.**

After the system functionality and time constraints are defined, the computational entities are generated through automated tools or by a manual coding process. This refinement step produces a set of concurrent tasks.

The set of tasks is defined by a suitable SysML model. The signals exchanged among the functional subsystems are implemented by a set of messages, defined in the same mapping model. Fig. 5 shows a (non-SysML) task model derived from the functional scheme of Fig. 3.

In a simple mapping implementation, every functional block generates a task which receives and sends messages to the tasks derived from the other functional blocks involved in the same control loop. If required, some tasks may be further refined into subtasks. For instance, the Lateral Camera task could be split in two tasks: one for frame capturing and one for image processing. The splitting increases the complexity of the scheduling problem, but could give benefits from the overall computational power utilization. Of course, the allocation of the two tasks in different computing units would result in a new message (frame) in the system.

The mapping model defines the refinement of the functional subsystems onto tasks and the task mapping onto the processing units of the execution architecture. Similarly, messages are associated with the interconnection buses. A feasible task allocation guarantees that all deadlines are met, that the traffic generated by tasks communication is schedulable and the semantics properties of the functional model are preserved. For this reason, the allocation of tasks should consider the computation and communication constraints at the same time.

## IV. Availability of the robotics testbench and next steps

The construction plans, including the components bill, the functional models, the software components and the platform and task models of the robotics testbench will be made available on a dedicated web site that aims at stimulating collaboration among researchers working in the fields of controls, modeling and real-time systems. We hope that the system architecture can provide a suitable benchmark and possibly a reference platform for teaching and student competitions.

As for future steps, besides the obvious target of completing the modeling effort and developing the missing tool components, we plan to explore the automated/optimal mapping of functional models onto a software/hardware architecture (e.g., as shown by Zheng et al. [9]; the end-to-end design of distributed real–time systems, where the different tasks communicate through shared memory or CAN Bus; the use of specification languages and planning for autonomous robots; and possibly resource aware control.

## References


[1] The MathWorks, Inc. Simulink: Simulation and Model-Based Design. [Online]. Available: http://www.mathworks.com/products/simulink/

[2] M. Di Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of multi-task implementations of simulink models with minimum delays," vol. 6 No 4, 2010.

[3] A. Sindico, M. Di Natale, and A. Sangiovanni-Vincentelli, "An industrial application of a system engineering process integrating model-driven architecture and model based design."

[4] OMG: Object Management Group. Modeling analysis of real time embedded systems (marte) profile. [Online]. Available: http://www.omg.org/spec/MARTE

[5] H. Bruyninckx. Open RObot COntrol Software. [Online]. Available: http://www.orocos.org/

[6] F. Moro, D. Fontanelli, and L. Palopoli, "Vision-based robust localization for vehicles," in *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*. IEEE, 2012, pp. 553–558.

[7] D. Nistér, "Preemptive ransac for live structure and motion estimation," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 199–206.

[8] D. Fontanelli, L. Palopoli, and L. Greco, "Deterministic and stochastic qos provision for real-time control systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*. IEEE, 2011, pp. 103–112.

[9] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli, "Synthesis of task and message activation models in real-time distributed automotive systems," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 93–98.