SenseDSL: Automating the Integration of Sensors for MCU-based Robots and Cyber-Physical Systems

Christian Berger
Chalmers | University of Gothenburg, Sweden
Department of Computer Science and Engineering
christian.berger@chalmers.se

ABSTRACT

Context: Cyber-physical systems like robotic platforms use sensors to perceive their surroundings. Multiple sensors like infrared, ultrasonic, and incremental encoders help these robots to orient act safely in their environment. *Objective:* However, finding a valid setup how to connect all sensors to a microcontroller (MCU) *and* to configure the embedded operating system correctly is hard because a *constraint-satisfaction problem* (CSP) needs to be solved. Here, a domain-specific language and an accompanying workflow are of great help for users to focus on algorithms instead of solving the CSP over and over again whenever a sensor configuration is adapted. *Method:* In our previous work, we have focused on modeling the CSP of the aforementioned problem appropriately. In this work, we are presenting *SenseDSL*, a simple, intuitive, and compact DSL to describe *what* shall be perceived by an MCU but letting the *how* to the tooling that processes the DSL's instances. *Results:* The workflow is realized by using Xtext and Xtend to process DSL artifacts, Prolog to solve the CSP, and a lean layer on top of ChibiOS/RT to interface with the hardware and to enable code reuse for a large variety of MCUs. *Conclusion:* Features of an embedded system are described in a compact tabular representation in combination with a textual description of the desired sensors to be connected. The model transformation and code generation process verifies the setup, finds automatically the best connection plan for the given sensors, and generates the required code for ChibiOS/RT and an accompanying host application to read and handle the data.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Cyber-Physical Systems, Code Generation, Verification, CSP

1. INTRODUCTION

Experimental robotic platforms and cyber-physical systems utilize sensors to perceive their surroundings and actuators to interact with their context. Thus, simple components like alert systems for home automation to monitor unauthorized opening of windows for example as well as complex systems like self-driving cars can be realized [2].

While the complex behavior of such systems can be evaluated experimentally in virtual test environments [1, 6], realistic sensor data from the real context is necessary to test the system with unreliable data to increase its robustness. Nowadays, such experimental platforms can already be realized with affordable commercial-of-the-shelf components [3].

More sensors enable a better perception of the robot's surroundings. Thus, having multiple instances of the same sensor mounted at different positions of such a platform as well as different sensors to compensate for their respective drawbacks require a thorough setup of both, the actual connection of the particular sensor to the microcontroller (MCU), and the configuration of the embedded operating system (OS).

1.1 Problem Domain

However, connecting a set of sensors correctly to the available set of pins of an MCU while considering the respective pins' multiple usages according to its hardware specification is a *constraint-satisfaction problem* (CSP)–especially, when the best setup shall be found. In our previous works [11] and [12], we have conducted several experiments with Prolog and Alloy to find an appropriate model representing instances of the problem domain. We concluded that describing the CSP as Horn clauses in Prolog allows an intuitive and processable way to handle all 14,689,111 possible multiple pin usages of our STM32F4 Discovery Board that we are using for our self-driving miniature car platform [2] as depicted by Fig. 1.

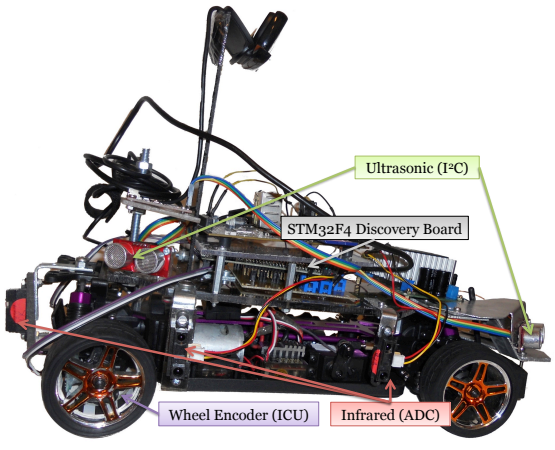

Figure 1: Miniature car platform using different sensors to perceive its surroundings.

1.2 Research Goal

Though, having a solution to the CSP still leaves the error-prone task of configuring the embedded OS correctly and to serialize/deserialize the data on changes of the sensor setup to the developer. When the MCU is simply used as the hardware/software interface, the software configuration of the embedded OS is merely about initializing the pins correctly, reading back the data regularly, and transferring it to a host system for further processing.

Therefore, the goal of this work is to present and describe an intuitive DSL and its accompanying tooling to complement our previous works and to provide a ready-to-use environment to the cyber-physical systems and robotics community. This DSL serves as the unifying environment for solving the CSP and for generating the appropriate code to use the sensors at hand for both, the embedded OS and the host, to which the MCU is connected.

1.3 Contributions of the Article

The tool-chain μCPSCodeGenerator combines the following four steps:

1. Processing formalized descriptions of an MCU's capabilities regarding the pins' multiple usages,
2. Verifying a requested pins' usage for a set of sensors and finding the best configuration concerning their lowest opportunity costs,
3. Generating the required code for the open source embedded realtime OS ChibiOS/RT [16] to interface with the given set of sensors, and
4. Generating an accompanying host application to receive the serialized data from the MCU and deserialize it properly for further handling; this application is realized in the multi-platform environment OpenDaVINCI [2].

The tool-chain and examples are made available as open source at http://www.christianberger.net/uproxy.

1.4 Structure of the Article

The rest of the paper is structured as follows: Sec. 2 describes related work. Sec. 3 presents the DSL and describes the integrated workflow to handle and verify the DSL's instances, to automatically find the best pin assignment, and to generate the code for ChibiOS/RT. Sec. 4 concludes the work and gives and outlook to future work.

2. RELATED WORK

Ghezzi et al. [7] as well as Gupta and Pontelli [8] have contributed to the problem of applying model-checking for formalized specifications. They also investigated the use of constraint logic programming to address the combinatorial nature; however, in contrast to our work [11], they did not specifically focus on finding the best solution to a given CSP.

Joshi et al. [9] also presented an approach to utilize a logic programming language to solve the pin assignment problem. However, their contributions describe rather a concept and do not include any practical case study or evaluation.

Tooling is also available from embedded system suppliers. The tool "STMCubeMX" [17] provides a graphical configuration environment to support the developer in setting up the proprietary STM library to interface with the MCU. Additionally, it identifies conflicting configurations and checks for realizability with the selected board. In contrast to our work, their tool apparently only supports the STM-supplied MCUs and their own proprietary library. Thus, the generated software cannot be used with other systems; furthermore, automatically finding the best pin assignment is not possible. Our tool-chain bases on the open source realtime OS ChibiOS/RT and hence, supports a large variety of MCUs.

The freely available tool environment CoSmart [5] aims for supporting similar use cases as the previously described commercial tool. However, their tool only visualizes conflicting pin configurations but does not support automated pin assignment and setup.

Lange et al. present in their work [10] a vision to combine high-level software components communicating by using the publisher/subscriber pattern from the Robot Operating System (ROS) [14] with low-level control algorithms running on FPGAs. They investigate how ROS nodes can be transparently migrated to an FPGA-architecture. Contrary to this work, their work does not encompass the automated verification and assignment of suitable pins with generating the accompanying code to handle the specific sensors.

The works presented by Zoppi [19] and Migliavacca et al. [13] aim for bringing ROS to the MCU world. Thus, such resource-constrained systems would be able to directly join a ROS network providing the data of interest. However, their so called μROSnodes do not provide specific middleware support to interface with sensors and actuators; in contrary, this is left to the lower layers of ChibiOS/RT and thus, needs to be realized correctly by the developer. Here, our work fills this gap by automatically generating valid code that is required to handle the sensors.

Resource-constrained devices are also addressed by Schwitzer and Popa. They use ARM Cortex-M3 MCUs from an energy-grid project as examples, for which they ported Google Protobuf [15]. Thus, data structures that are exchanged between a host and an MCU can be easily described and reused. In contrast to their work, we are focusing on an integrated and automated approach by finding a valid pin assignment, deriving the required data structure, and generating the required code to configure and handle the sensors, as well as to exchange the data between the MCU and a host.

The integrated workbench "mbeddr" is outlined by Voelter [18]. The work aims for supporting the entire development lifecycle for embedded software by formalizing and integrating all necessary artifacts ranging from specifications, state machines, C-code, test cases, and documentation. At the time of writing, an approach for computing the best way to connect sensors to an MCU is not supported and left to DSL engineers to extend the existing language workbench instead.

3. SENSEDSL AND THE MODEL-TO-CODE TRANSFORMATION WORKFLOW

An MCU is used to interface with a cyber-physical system's context by regularly reading the data from its connected sensors. Afterwards, the data is either directly processed on the MCU to interact with the surroundings by using attached actuators, or transferred to a more powerful host system to further process the data.

3.1 Describing the MCU

Developers of an MCU start with a pin configuration data sheet like it is exemplarily depicted as an excerpt from the STM32F4 Discovery Board in Tab. 1. The task is to find a pin assignment for a given set of sensors and actuators so that all sensors can be read regularly and all actuators can be accessed by a control algorithm while fulfilling a predefined cycle time. Hereby, pins with multiple usages are limited resources and prevent their use for other or future purposes; in the aforementioned example, the use of pins PA2 and PA3 for analog input sensors would block their use for more input capturing units (ICUs).

Tab. 1 shows only five pins with multiple usage; the complete table has many more columns to support onboard devices of the STM32F4 Discovery board like a temperature and acceleration sensor, serial ports, CAN bus, Ethernet pins, PWM pins, and more. Thus, finding an appropriate pin connection setup is time-consuming and identifying the best connection setup in terms of lowest opportunity costs (i.e. using only pins with less multiple usage capabilities) is a hard task and hence, a CSP.

[1]In the following, ADC is used to refer to analog sensors.

Pin	ADC[1]	I^2C	ICU
PA1	ADC123-IN1	-	TIM2_CH2
			TIM5_CH2
PA2	ADC123-IN2	-	TIM9_CH1
PA3	ADC123-IN3	-	TIM9_CH2
PA8	-	I^2C3-SCL	-
PB0	ADC12-IN8	-	-
PB1	ADC12-IN9	-	-
PB4	-	-	TIM3_CH1
PB5	-	-	TIM3_CH2
PB6	-	I^2C1-SCL	-
PB7	-	I^2C1-SDA	TIM4_CH2
PB8	-	I^2C1-SCL	TIM10_CH1
PB9	-	I^2C1-SDA	-
PB10	-	I^2C2-SCL	-
PB11	-	I^2C2-SDA	-
PB14	-	-	TIM12_CH1
PB15	-	-	TIM12_CH2
PC1	ADC123-IN11	-	-
PC2	ADC123-IN12	-	-
PC4	ADC12-IN14	-	-
PC5	ADC12-IN15	-	-
PC6	-	-	TIM3_CH1
			TIM8_CH1
PC9	-	I^2C3-SDA	-
PE5	-	-	TIM9_CH1
PE6	-	-	TIM9_CH2
PE9	-	-	TIM1_CH1
PE11	-	-	TIM1_CH2

Table 1: Excerpt from possible multiple pin configurations of the STM32F4 Discovery Board.

In our previous work [11] and [12], we have described how to appropriately model the pin assignment problem as a CSP in Prolog and Alloy; in our DSL processing workflow, we are using Prolog as the problem formulation is more intuitive in contrast to Alloy where counter-examples are used and thus, the problem representation needs to be inverted. Furthermore, our experiments showed that Prolog performs better for our CSP.

In our workflow, we preserved the tabular representation for the developer who can specify the properties of the pins by using an ordinary spreadsheet tool. This task needs to be carried out once per board.

3.2 Describing the Desired Sensor Connections

Next, the developer needs to define *what* shall be connected to the MCU. Therefore, we have defined a simple, compact, and intuitive language called *SenseDSL* in Xtext. A concrete example is shown in Fig. 2.

The core of the language is the specification of a list of sensors, their return types, and their connection properties. In our case, we focus on analog inputs as measured voltage levels, I^2C serial bus, and input capturing units (ICUs). Thus, we support analog infrared sensors, digital ultrasonic sensors, and incremental encoders as found in experimental robotic platforms as depicted in Fig. 1.

The list of sensors is embedded into an encapsulating scope `board`; currently, this structural element serves for readability purposes. Furthermore, the cycle time for each sensor can be specified (otherwise, 1,000ms is used during code generation)

```
board STM32F4DiscoveryBoard {
    sensor IRFront returns uint16 {
        id = 17;
        type = analog;  // infrared
    }
    sensor IRFrontRight returns uint16 {
        id = 18;
        type = analog;
        delay = 60;
    }
    sensor USRear returns uint8 {
        id = 19;
        type = i2c;     // ultrasonic
        address = 0xE6;
        delay = 40;
        prio = +2;
    }
    sensor WheelEncoder returns uint32 {
        id = 20;
        type = icu;
    }
}
```

Figure 2: Instance of *SenseDSL*.

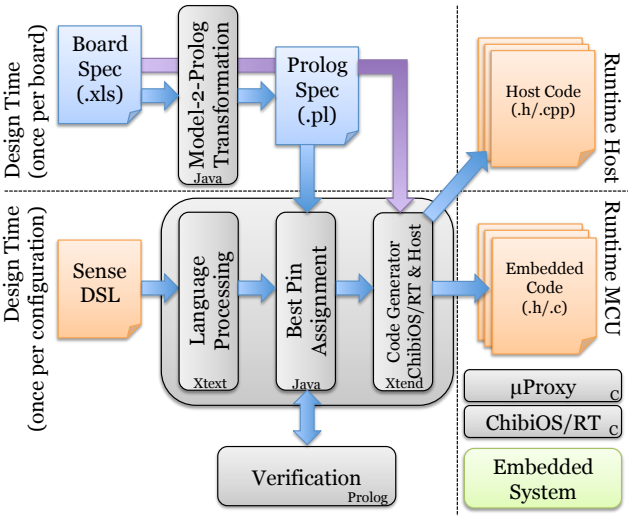

Figure 3: Workflow to transform an instance of *SenseDSL* and an MCU board description into executable code for ChibiOS/RT and accompanying host code: Firstly, the tabular board description is transformed into a Prolog representation for the constraint-satisfaction problem (CSP); next, an instance of *SenseDSL* is processed in Xtext/EMF. On success, sensor properties are extracted and used to verify the desired configuration and to find the best solution for the CSP with Prolog. If a solution has been found, the appropriate code for ChibiOS/RT is generated that is used in our µProxy layer on top of ChibiOS/RT to handle the sensors; additionally, code for the host is generated, to which the MCU is connected to receive and handle the serialized data.

and a priority level for the task scheduling in ChibiOS/RT for each sensor class; both parameters are not used for ICUs as they are triggered automatically by ChibiOS/RT whenever there is a changing edge detected on the specified input pin of the MCU.

In our workflow, an instance of *SenseDSL* needs to be specified and updated whenever the sensor configuration is adapted, i.e. adding or removing a sensor, changing the return types or identifiers, or modifying scheduling properties.

3.3 Model-to-Code Transformation

Both aforementioned artifacts are used during their processing in our workflow. Its overview is depicted in Fig. 3. On overview which element of the language *SenseDSL* is used for which part of the workflow during the verification and code generation is shown in Tab. 2.

Firstly, the tabular representation of the MCU hardware specification is transformed into a Prolog model so that the CSP can be solved for a given instance of *SenseDSL*. This transformation is conducted as described in [11]; to find the best solution in terms of lowest opportunity costs, the number of multiple usages per pin is extracted so that tuples with pins are preferred whose summarized multiple pin usages is the smallest. This transformation process is required only once per MCU.

Next, a given instance of *SenseDSL* is read by Xtext/EMF to find syntactic mistakes and to generate an AST on success, which is later processed during the model transformation. Subsequently, `sensor` elements are traversed with Xtend to collect the specified connection `type`s into a tuple that is used to query the board model representation in Prolog. Firstly, the query checks whether a pin assignment can be found as a solution to the CSP; afterwards, the solution with the lowest opportunity costs is determined and returned to the Xtend transformation process.

Finally, the code generation is carried out using the results from the verification and pin assignment step. Therefore, further information from the MCU board description is needed to finally generate the source code for ChibiOS/RT as depicted by the purple arrow in Fig. 3. This includes, for example, which driver shall be used to process analog inputs (ADC1, ADC2, or ADC3) and I^2C (I^2C1, I^2C2, or I^2C3), or which channel is required to capture the input of ICUs. Next, the sensors for the same driver are merged and the corresponding ChibiOS/RT C-code to configure and regularly query the respective sensors is generated.

DSL Element		Verification & Pin Assignment	Code Generation during the *SenseDSL* Workflow							
			ChibiOS/RT							Host Application
			Sensor				Sensor Data		Task Scheduling	
			Configuration			Reading	Storing	Serialization		Deserialization
			ADC	I^2C	ICU					
	board's name	○	○	○	○	○	○	○	○	○
sensor attribute	name	○	○	○	●	●	●	●	○	●
	return type	○	○	○	○	●	●	●	○	●
	identifier	○	○	○	○	○	○	●	○	●
	type	●	●	●	●	●	○	○	●	○
	address	○	○	●	○	●	○	○	○	○
	delay	○	○	○	○	○	○	○	●	○
	prio	○	○	○	○	○	○	○	●	○
	inferred pin	○	●	●	●	○	○	○	○	○

Table 2: Overview of the use of elements from *SenseDSL* during the different phases of the workflow.

3.4 μProxy Runtime Environment

The generated C-code is supposed to be used with μProxy, which is a compact software layer that is running on top of ChibiOS/RT. This layer includes a ChibiOS/RT user shell to interactively communicate with the board during the development to print out the raw data that is perceived from the sensors. Furthermore, it adds a data exchange protocol for the communication between the host and the MCU; in the case of the STM32F4DiscoveryBoard, the data is sent using the Serial-over-USB connection allowing the host to read the data from the `/dev/ttyACM0` device node on Linux for example.

Therefore, the perceived data from the different sensors is collected and transformed into a compact Netstrings representation [4] by using the identifiers as specified in *SenseDSL* to distinguish the different data sources. The identifier itself is encoded as a `uint8` type to keep the transferred bytes compact. The example from Fig. 2 is encoded using 17 bytes in total including 4 bytes overhead for Netstrings.

3.5 Code to Receive and Handle Data on a Host

As the last step, μCPSCodeGenerator generates a C++ application for the multi-platform middleware OpenDaVINCI[2] to provide exemplary template code to further process the data perceived by the MCU on a host computer. Therefore, the application uses the incoming data from the device node as data trigger event source and parses the Netstrings. Whenever a complete Netstrings has been successfully received, the contained data is deserialized into a corresponding data structure by using the sensor identifiers and return types as described in the instance of *SenseDSL* by the developer.

With *SenseDSL* and the accompanying workflow as realized in μCPSCodeGenerator, a complete and integrated environment is available. Properties of the equipment like sensors and MCUs at hand can be easily described to finally and automatically end up with valid code for an embedded realtime OS reflecting the configuration with the lowest opportunity costs. Furthermore, the tool-chain also generates a data exchange protocol and the necessary serialization and deserialization methods to further process the data on a more powerful host computer as required different cyber-physical systems or robotic applications.

4. CONCLUSION AND OUTLOOK

Planning and realizing a valid or especially the best pin assignment for an MCU from a given set of sensors is a time-consuming and error-prone task. The reason is that developers need to solve a constraint-satisfaction problem (CSP) regarding potential multiple pin usages enabled by the MCU. Thus, finding an assignment with the lowest opportunity costs to enable further or future use cases for the MCU is difficult.

However, solving the CSP can be carried out by a computer when it is modeled properly. In our previous work, we have investigated different approaches and came to the conclusion that using Horn clauses in Prolog offers an intuitive and efficient way to get valid solutions for practical CSPs.

[2]www.christianberger.net/opendavinci

In this work, we have presented *SenseDSL* and its accompanying workflow, which combines the CSP solving approach with an intuitive and compact way to describe a given set of sensors that shall be realized by an MCU. Therefore, a developer firstly outlines the capabilities of the MCU's pins as a tabular representation that is transformed into a Prolog model; next, the set of sensors and their respective connection requirements like analog or I^2C are described. The model processing workflow utilizes both artifacts to firstly verify that the CSP is solvable before a solution with the lowest opportunity costs is determined. Afterwards, the solution is used to automatically generate code for the realtime embedded operating system ChibiOS/RT to interface with the sensors and to send the data to the host. Finally, μCPSCodeGenerator generates the accompanying host application that receives the data from the MCU to complete the data capturing workflow.

Future work is required to complement *SenseDSL* by language constructs that allow the use of actuators like PWM-based motors for example. Furthermore, additional constraints that need to be considered during the CSP solving process could be included to describe mounting-related restrictions to exclude or prefer a specific side of an embedded board for example. Furthermore, the problem of having several MCUs and different sensors at hand to be connected can be addressed to find the optimum solution with the overall lowest opportunity costs.

The software environment as described in this article is made available at http://www.christianberger.net/uproxy as open source to serve the cyber-physical systems and robotics community.

5. REFERENCES

[1] C. Berger. *Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles*. Shaker Verlag, Aachener Informatik-Berichte, Software Engineering Band 6, Aachen, Germany, 2010.

[2] C. Berger. From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation. *Journal of Software Engineering for Robotics*, 5(1):63–79, June 2014.

[3] C. Berger, M. A. Al Mamun, and J. Hansson. COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle. In E. M. Schiller and H. Lönn, editors, *Proceedings of the 2nd Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS)*, page 12, Toulouse, France, Sept. 2013.

[4] D. J. Bernstein. Netstrings. 1997.

[5] CooCox. CooCox CoSmart.

[6] B. P. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, 2003.

[7] C. Ghezzi, D. Mandrioli, and A. Morzenti. TRIO : A Logic Language for Executable Specifications of Real-Time Systems. *Journal of Systems and Software*, 12(2):107–123, May 1990.

[8] G. Gupta and E. Pontelli. A Constraint-based Approach for Specification and Verification of Real-time Systems. In *Proceedings Real-Time Systems Symposium*, pages 230–239. IEEE Comput. Soc, Dec. 1997.

[9] B. Joshi, F. M. Rizwan, and R. Shettar. MICROCONTROLLER PIN CONFIGURATION TOOL. *International Journal on Computer Science and Engineering*, 4(05):886–891, 2012.

[10] A. B. Lange, U. P. Schultz, and A. S. Soerensen. Towards Automatic Migration of ROS Components from Software to Hardware. In C. Schlegel, U. P. Schultz, and S. Stinckwich, editors, *4th International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob)*, pages 22–25, Tokyo, Japan, Nov. 2013.

[11] M. A. A. Mamun, C. Berger, and J. Hansson. Engineering the Hardware/Software Interface for Robotic Platforms - A Comparison of Applied Model Checking with Prolog and Alloy. In C. Schlegel, U. P. Schultz, and S. Stinckwich, editors, *4th International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob)*, pages 26–34, Tokyo, Japan, Nov. 2013.

[12] M. A. A. Mamun, C. Berger, and J. Hansson. MDE-based sensor management and verification for a self-driving miniature vehicle. In J. Gray, S. Kelly, and J. Sprinkle, editors, *Proceedings of the 2013 ACM Workshop on Domain-Specific Modeling (DSM)*, pages 1–6, Indianapolis, IN, USA, Oct. 2013. ACM.

[13] M. Migliavacca, A. Bonarini, and M. Matteucci. Modular Development of Mobile Robots with Open Source Hardware and Software Components. In S. Behnke, M. Veloso, A. Visser, and R. Xiong, editors, *RoboCup 2013: Robot World Cup XVII*, pages 576–583. Springer Berlin Heidelberg, 2014.

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS : an open-source Robot Operating System. In *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, number Figure 1, Kobe, Japan, May 2009.

[15] W. Schwitzer and V. Popa. Using Protocol Buffers for Resource-Constrained Distributed Embedded Systems. Technical report, Institut für Information, Munich, Germany, Nov. 2011.

[16] G. D. Sirio. ChibiOS/RT, Aug. 2014.

[17] STMicroelectronics. STM32CubeMX. Technical report, June 2014.

[18] M. Voelter. Preliminary Experience of using mbeddr for Developing Embedded Software. In H. Giese, M. Huhn, J. Philipps, and B. Schaetz, editors, *Proceedings of the Workshop "Model-based Engineering of Embedded Systems X"*, pages 73–82, Wadern, Germany, Mar. 2014.

[19] A. Zoppi. *A Lightweight Open-Source Communication Framework for Native Integration of Resource-Constrained Robotics Devices with ROS*. Master thesis, Politecnico Di Milano, 2012.