Measurement-based real-time analysis of robotic software architectures

Nicolas Gobillot, Fabrice Guet, David Doose, Christophe Grand, Charles Lesire, Luca Santinelli

Measurement-based real-time analysis of robotic software architectures

Nicolas Gobillot, Fabrice Guet, David Doose, Christophe Grand, Charles Lesire, Luca Santinelli

Abstract— **Providing guarantees on the system behavior is mandatory in order to let the robots enter our every-day life. Among these guarantees, proving the fulfillment of real-time constraints on the software is a key issue, as their violation could result into unexpected and unsafe behaviors. In this paper, we present a methodology to guarantee real-time constraints on component-based software architectures of robots. This methodology relies on the MAUVE language to model the component architecture, and on a set of analysis tools that first estimate the worst case execution time of elementary functions from actual component traces, and then check the real-time constraints of each component. We illustrate this process on the architecture developed for the autonomous navigation of a partially known area by a mobile robot.**

I. INTRODUCTION

Works that develop or evaluate advanced functions for autonomous robots flourish. They propose advanced control techniques, environment reconstruction, navigation in complex, unknown, dynamic environments, interaction with human-beings, etc. In order to have all these techniques be used on actual robots in interaction with their environment, we must be able to guarantee their safe and correct behavior. This consideration has already been a major concern of roboticists for several years. Timing properties are crucial for the safety of the system as they could impact system execution and its functional performance. It is for instance proved that theoretically sound control techniques may become hazardous when facing timing disturbances [1], [2]. In this work we refer to a robotic system as a real-time system, where real-time systems are computing systems which must react within precise time constraints to events in the environment. Real-time systems are often called predictable or deterministic systems, since they demand for predictable and always reproducible behavior.

A. Real-time applications in robotics

Proving the real-time execution of applications has not been of major concern for roboticists, as they have mainly focused on algorithmic performance. Most of the works have considered that missing some deadlines would have no significant impact on the application. They have only considered making some measures of computation time of the functions [3], [4], [5], [6] or response time over the network for distributed algorithms [7] for few runs, in order to evaluate timing properties of the systems. If the system is not strictly real-time, such measurements can suffice for having confidence that the algorithms can be embedded.

Authors are with ONERA – The French Aerospace Lab, Toulouse, France
firstname.lastname@onera.fr

The lack of interest from roboticists for proving real-time constraints also comes from the applicability of real-time analyses. Indeed, robotics system are characterized by their modularity, the complexity of the algorithms and the diverse tasks executions models. On the other hand, real-time analysis methods need predictability and determinism for being applied. A way to obtain that is the use of the Worst-Case Execution Time (WCET), which is defined as the worst-possible execution time for a task. The WCET is a single value which is assumed as the execution time of the task in all its execution instances. The safety of real-time systems is assured once the timing constraints are respected.

B. Related works

Best practices towards real-time analysis of robotic systems have relied on model-based approaches. It is indeed necessary to have a model of the software running on the system in order to be able to check that real-time constraints will be ensured, depending on the scheduling policy of the system, and on the tasks properties (periods, deadlines, priorities). In [8], the authors make an AADL model of their architecture, including hardware models, and analyse the latency of several components. However, they do not state how they get the worst case computation time of some functions. Moreover, their model is made *a posteriori*, then making the result questionable with respect to what is really implemented on the system. The SmartSoft framework [9] provides tools for describing components and how they are mapped to real-time tasks of the system. They eventually use the Cheddar tool [10] to simulate the scheduling of tasks in order to evaluate the schedulability of the architecture. However, they do no state how they get the WCET of tasks nor give any hint on how they could be computed.

C. Contribution

In this paper, we present an approach for evaluating the real-time execution of a robotic application that tends to be less pessimistic than classical real-time approaches and then more applicable to robotic systems.

This approach combines existing works into a global process for real-time analysis. The quality improvement of the analysis relies on two major points:

- We use a more detailed model of the task behavior; for that purpose, we rely on component-based models using the MAUVE language [11], and on a schedulability analysis that takes such models into account [12].
- We estimate WCET of tasks from actual measurements, then taking into account potential influence from input data and system interactions.

The approach is presented in Section II with a brief description of existing works that have been integrated. In the same section it is proposed a discussion on the way execution traces are obtained and applied. Section III then presents a case study in which this process is applied to the software architecture of a mobile robot performing an exploration mission.

II. REAL-TIME VALIDATION PROCESS

The real-time validation process presented in this paper is summarized on Fig. 1. First, it is necessary to make a model of the several components of the software architecture, and then to specify how they will be deployed on the system. The analysis needs to know the WCET of the elementary functions executed by the architecture which depend on the target platform. To obtain these WCET, the process is to generate some instrumented code and execute it on the actual target platform. From the traces obtained during this execution, it is possible to have a statistically sound evaluation of the WCET. From the models and these WCET, it is now possible to compute the Worst-Case Response Time (WCRT) of the several components in order to conclude on the architecture schedulability.

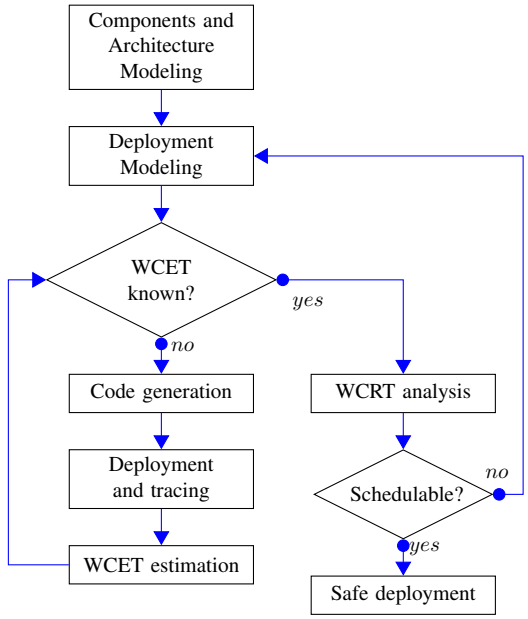

Fig. 1: Real-time analysis process

Each step of this process is further explained in this section.

A. Component and Architecture Modeling

The modeling is done using a robotic, component-based Domain Specific Language, MAUVE [11]. A MAUVE model of a software architecture is made of 4 nested levels:

- *codels*, which are *elementary codes*, i.e. functions implemented in C or C++ language and which implement the algorithmic part of the architecture; only codels signature is defined in the MAUVE DSL;
- *components*, composed of a *shell* describing their interface (input and output data ports, provided or required operations) and a *core* describing their behavior by a finite state machine; in each state, it is possible to call a codel or to access to ports (reading/writing) or operations;
- *architectures*, in which components are instantiated and connected one to the other;
- *deployments*, which indicate how the components of an architecture are mapped to real-time tasks, by defining their periods, priorities, deadlines and affinities (which CPU core they will run on).

Listing 1 presents a snapshot of the Mauve model of a path planning component. Its shell is defined by a parameter (the path resolution), input ports (current pose of the robot, goal pose, current map), and an output port (the computed path). The core is defined by a state machine containing the *Idle* state (waiting for a new goal to arrive) and the *Planning* state. In that state, the component calls the *astar* codel.

Listing 1: Mauve code of a `path planning` component

```
shell PathPlanningShell {
  property resolution: double = 0.3
  input port map: OccupancyGrid
  input port pose: PoseStamped
  input port goal: PoseStamped
  output port path: Path
}

codel astar(in p: PoseStamped, in goal: PoseStamped,
     in map: OccupancyGrid): Path

core PathPlanningCore (PathPlanningShell) {
  var pose_: PoseStamped
  var goal_: PoseStamped
  var map_: OccupancyGrid
  var path_: Path
  var new_goal: bool = false

  statemachine {
    initial state Idle {
      run = {
        if (read(goal, goal_) == NewData) then {
          new_goal = new_plan(goal, pose);
        }
      }
      transition if (new_goal) select Planning
    }
    state Planning {
      entry {
        read(pose, pose_); read(map, map_);
        path_ = astar(pose_, goal_, map_);
        write(path, path_);
      }
      transition select Idle
}}}
```

As the codels contain the algorithmic part of the components, the execution time of the tasks mainly rely on the execution time of the codels. It's also important to notice the model of the architecture is used in both the code generation and the schedulability analysis. Consequently, both the real execution and the analysis are based on the same formal specification.

B. Code generation and tracing

In order to do the schedulability analysis, we need to obtain timed execution traces of our components. From the MAUVE models of components and architecture, embedded

executable code is generated using the Orocos [13] middleware. Each instance of a component is a real-time task, where the behavior follows the state-machine description, and that calls manually defined codels. Architectures and deployments are plugged into Orocos deployment scripts which instantiate components, connect them together and define their activities.

Along with the component code, the code behavior is traced using the LTTng framework [14]. This framework has been chosen as it is known to have a low impact on the execution behavior[1]. These traces are generated within the components code so that we can log each important part of the architecture. The resulting trace provides:

- call time of codels, and the caller component name;
- return time of the same codels;
- start and end times of each component execution cycle;
- entry and exit times of each state of each component.

From the timing data of these traces it is possible to extract the execution time of each codel execution by removing the interactions (preemptions or delays) of the more prioritary tasks. At the end of this step we obtain for each codel a sequence of execution times. It is then possible to get measures from several runs, where codels are executed, but not necessarily with the same architecture or in the same condition.

C. WCET estimation using the Extreme Value Theory

The static timing analysis [15] requires a precise model of the hardware as well as a formal representation of the tasks to estimate WCETs. Unfortunately, some hardware used in robotics systems, such as commercial off-the-shelf platforms, are not well documented and cannot be formalized precisely. Moreover, some complex algorithm like SLAM or planning leads to pessimistic WCET representations.

Probabilistic timing analysis is emerging as alternative to the static timing analysis. As it is a probabilistic approach, it is the notion of probabilistic Worst-Case Execution Time (pWCET) that has to be considered. The pWCET is the worst distribution of execution times, where each value C_i is associated with a probability p_i; p_i is the probability to exceed C_i. Probabilistic approaches, also called Measurement-Based Probabilistic Timing Analysis (MBPTA), only need measurements of execution time for inferring the pWCETs estimates. These approaches combine measurements and probabilistic analysis for WCET estimation [16].

MBPTA approaches consist of two main steps: i) the collection of *measurements of task execution time in real execution conditions as traces*, and ii) the *probabilistic analysis for inferring the pWCET estimation*. The measurements are important for extracting observable features such as average behaviors and trends that can appear while executing tasks. In order to obtain the pWCET from the measurements, the probabilistic analysis is based on the Extreme Value Theory (EVT), allowing to infer the rare events (where the worst-cases should be), very costly to measure. Doing this, the analysis provides safe pWCET estimates by mathematically inferring larger execution times than those measured. By safe estimates we mean that the pWCET is larger than or equal to the exact (and unknown) pWCET distribution, and to any possible empirical execution time distribution from the measurements.

Theoretically, the EVT is established for independent and identically distributed (i.i.d.) measurements [16], but in practice the less constraining hypotheses of stationarity and extreme independent measure(ment)s are enough to apply the EVT [17], [18]. Thus, the EVT can be reliably applied to realistic cases (non time-randomized systems) with certain degrees of dependences between executions, such as the estimation of the WCET of codels in our component-based architecture. The safety of the pWCET estimates with EVT comes from the respect of the hypotheses, [17], [18], [16].

The MBPTA is able to provide two main results: i) the WCET of each codels (WCET estimate) and ii) the WCET estimate reliability. While the WCET of the codels is used for schedulability analysis, the WCET estimate reliability is used to characterize the applicability of the MBPTA.

a) WCET estimates: A pWCET estimate is inferred from a trace of measurements providing tuples associating an execution time C to a probability p giving the confidence in not exceeding C. For instance, the confidence could be chosen to $p = 10^{-7}$, allowing to have sufficient confidence $(1 - 10^{-7})$ on the WCET model C.

b) WCET estimate reliability: The MBPTA tool we have developed is conceived as a logical workflow checking the applicability of the EVT with specific tests [18]. For an input trace of execution times it is possible to provide a pWCET estimate and an associated reliability with regard to the EVT applicability hypotheses. In particular, the reliability qualifies the traces of measurements (e.g. lack of information, verification of certain statistical properties,) and defines the quality of the pWCET estimates in representing the worst-case execution behavior of the codels. The reliability accounts for the safe inference of the pWCET with the EVT indicating that resulting WCET estimates have a sense (if they do not diverge). Based on this reliability, we can decide whether more timing samples must be gathered and analysed or not.

D. Schedulability analysis

In order to check the schedulability of the software architecture, we need to compute the Worst Case Response Time (WCRT) of the architecture's components. The Response time correspond to the time between the triggering of the task and the end of its execution, including tasks preemptions and delays. A component is schedulable if its WCRT is lower than its deadline. Obviously, a complete system is schedulable if all the components are schedulable.

Classical schedulability analysis [19] deals with monolithic task models, where each component of our architecture would have a unique WCET. Recent WCRT approaches [20], [21], [12] have used more detailed task models to improve the quality of the WCRT analysis. [12] models the tasks

[1] `http://lttng.org/features/#high-performance`

as Periodic State Machines (PSM), where each PSM fires a transition at each execution cycle, and each transition has a specific WCET. We have then developed a transformation from the MAUVE models of components to PSM in order to have the WCRT analysis take into account the state machines of our components. This transformation is only used for the analysis, the executed code being generated from the initial state machine description.

III. EXPLORATION BY A MOBILE ROBOT

The real-time analysis process has been used on a software architecture of a real robot performing an indoor navigation mission. The aim of this experiment is to autonomously and safely join some waypoints in a partially known environment while updating a 2D map and avoiding (dynamic) obstacles. The robot used in these experiments is a two-wheeled Pioneer 3DX mobile robot from Adept Mobile Robots. It is equipped with an Hokuyo UTM-30LX laser range finder.

This experiment requires the following functionalities that need to be executed conjointly. First, a localization and mapping algorithm (SLAM) is used to improve the robot odometry and build a map based on occupancy grid model. Functions dealing with motion control and obstacle avoidance are also implemented. For each target waypoint, the navigation and the guidance functions compute the path to reach this target point and manage the robot motion orders required to follow this path. The last functions are associated to the system hardware, particularly the drivers of the laser range finder (Hokuyo) and the mobile robot (P3DX) which allows to control the robot motions and provides the odometry estimate based on encoders measurement.

A. Components and architecture models

Each system functionality is implemented in different components using the MAUVE DSL. Figure 2 gives an overview of the system architecture. The low-level functions (drivers, in orange) are in the following components: *hokuyo* is the laser driver that gives periodically a laser scanning; *p3dx_driver* is the robot driver that takes linear and angular velocities as input and provides, as output, the robot pose computed from the encoders. The robot is controlled by a set of components (*guidance*, *control* and *safety_switch*, in blue) in order to follow the path computed by the planning components (in green). A command from the operator can also arrive from the *teleop* component.

The navigation and SLAM functions are dispatched onto several components (in green): one is a wrapper of the GMapping algorithm [22]; *pose* computes a pose from the robot odometry and the SLAM correction; *navigation* provides functions to first compute a path to join a target waypoint, based on the classical A* algorithm, and second send successive points of the path to the guidance component.

The *safety_switch* has two functions: first it is a safety component that reduces or sets to zero the robot's velocity control if an obstacle is detected in front of it, and second can switch between the teleop command and the control command.

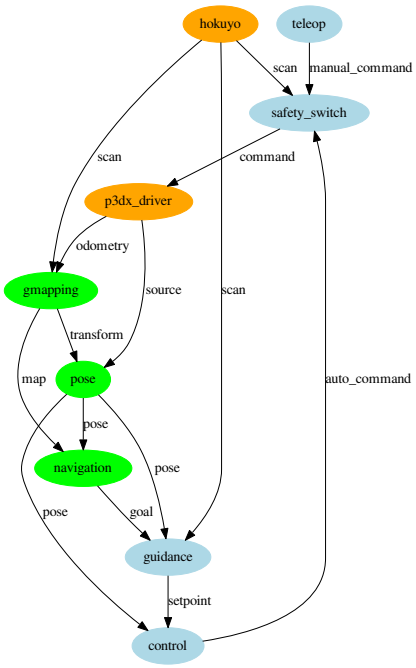

Fig. 2: Navigation component-based architecture. Ovals represent components. Edges represent port connections between components.

B. pWCET estimation

From the models of the above-mentioned architecture, we have generated the components' code and deployed these components to have some execution traces. The target platform we use on our Pioneer robot is a Intel i5 laptop with a 4-core CPU at 2.7GHz, running Ubuntu 14.04 configured to use the real-time `SCHED_FIFO` Linux scheduler.

The pWCET estimation uses measurements of the execution time of elementary functions (*codels*). The estimation is independent of the complete software architecture running on the system, and consequently it is possible to make several runs with different architectures, and then combine the measurements for the pWCET estimation. The runs on our platform have then be split into:

- automatic control runs, in which we only deployed the control-related components; we manually sent some guidance goals to the guidance components, for several path lengths and on several environments (typically with/without obstacles) to have a good variability of conditions;
- SLAM runs, in which we only deployed the SLAM-related components; we manually drove the robot while building a map, performing different loops over the environment, with several paths;
- navigation runs, in which we only deployed the navigation components with a static global map from the previous run, without moving the robot: the navigation component has been asked to compute paths with random objectives.

We incrementally performed all these runs by checking the pWCET estimate reliability criteria. Once the criteria are good enough, we stopped making more runs, and the computed pWCET estimates were considered reliable enough for the sequel of the schedulability analysis. An example of the pWCET estimate is given in Fig. 3 and 4. Figure 3 shows the measured execution times for the `avoid_collision` codel of component *guidance*.

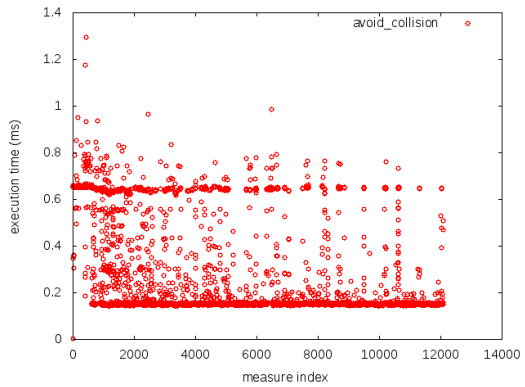

Fig. 3: Execution time measures for codel `avoid_collision`. Each circle represents a measure.

Figure 4 shows the result of the pWCET estimation. The inverse cumulative probability distribution of measures is represented by red circles. The pWCET estimates is given by the green curve. This curve gives estimation of the WCET of the codel for the desired probability threshold.

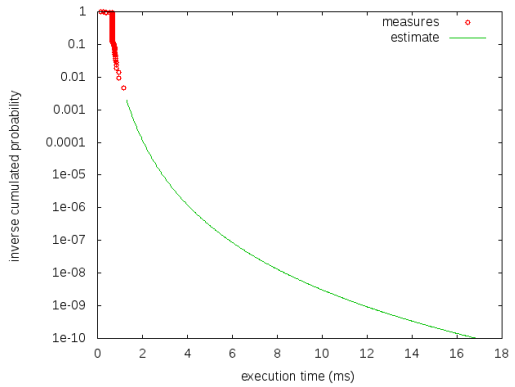

Fig. 4: pWCET estimate for codel `avoid_collision`

C. Schedulability analysis

The schedulability analysis then uses the components and architecture models, as well as an estimation of the WCET of all the codels called in the components. The deployment specification is given in Tab. I. We have set the period of almost all components to 100ms., while time-consuming components, namely *gmapping* and *navigation* are scheduled at a period of 1s. All deadlines are equal to periods. The priorities have been defined so that the drivers are the components with the highest priority, in order to not miss data. Then the components that manage the safety and integrity of the platform have the highest priorities. Finally, the lowest priorities have been set to the less important components (with respect to robot safety). The affinity, i.e. the core on which the components' tasks will run, are set to reserve one core to the gmapping component, which is highly time consuming, while other components run on a specific core.

TABLE I: Deployment specification

component	period (ms)	deadline (ms)	priority	affinity
p3dx_driver	100	100	10	1
hokuyo	100	100	9	1
safety_switch	100	100	8	1
pose	100	100	7	1
guidance	100	100	6	1
control	100	100	5	1
teleop	100	100	2	4
navigation	1000	1000	2	1
gmapping	1000	1000	3	2

In order to apply the schedulability analysis method, we first need to estimate WCET values. From the pWCET computations, we have to select the probability threshold p. Tables II and III present the several values of the WCRT of each component, depending on the probability threshold p. We remind the reader that p reprensents the confidence we can have that the actual WCET will not exceed the estimated value. We have made p vary from 10^{-3} to 10^{-9}, which are common values used in safety assessment of critical systems.

TABLE II: WCRT results for components running on core 1. Values are in milliseconds. Values in bold are exceeding the component deadline.

component	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
p3dx_driver	8	10	11	13	17	29	55
hokuyo	28	32	33	36	40	52	78
safety_switch	31	35	36	41	46	61	90
pose	32	36	37	42	47	62	91
guidance	35	39	40	45	50	65	94
control	38	42	43	48	53	68	98
teleop	45	49	50	55	60	75	**499**
navigation	764	817	841	876	906	984	**1146**

TABLE III: WCRT results for components running on core 2. Values are in milliseconds. Values in bold are exceeding the component deadline.

component	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
gmapping	703	711	715	722	764	994	**2258**

We can first notice that the WCRT are logically increasing when we reduce p, as we want to be more and more confident in the WCET estimates. Moreover, for components running on core 1, we can notice that all components are schedulable down to confidence 10^{-8}. For a confidence of 10^{-9}, only the *teleop* and *navigation* components are not schedulable. A

similar behavior can be observed for the unique component running on core 2: *gmapping* is schedulable for all chosen values of p, but for 10^{-9}. A discussion on the impacts of the results on the architecture design is discussed hereafter.

D. From WCRT to architecture design

The WCRT analysis proves that the real-time constraints of all components will be respected at execution with a confidence at least of 10^{-8}. In the case we need a higher confidence (i.e. $p \leq 10^{-9}$), the WCRT analysis results will help us designing our architecture. For instance, regarding the *gmapping* component, two solutions are possible without changing the component code:

- increase the component period (e.g. to 2.5 seconds), to have the component be schedulable with a high confidence, despite that the processing will be slower;
- make the component aperiodic, reacting to incoming data (scans and poses) when they arrive and when the component has time to process them; most of the time the processing will be faster, but a freeze of the processing is still possible.

As a conclusion, the WCRT analysis, will first give some confidence in the fulfillment or real-time constraints for time-critical components, and will also provide some information useful for designing an embedded robotic architecture.

IV. CONCLUSION

In this paper, we have presented a methodology for analyzing the real-time properties of a component-based software architecture. This methodology relies on MAUVE models of the architecture from which instrumented embedded code is generated. We are then able to estimate the pWCET of components from actual measurements. From these pWCET and the models, we applied a WCRT analysis that provides schedulability results with a given confidence. We have also discussed on a real application of the presented process can first give some confidence on the critical parts of the architecture, and insights for the design of the overall software architecture.

From the experimental results we have conducted, several future works have appeared. First, we must have a better control on the real-time execution of components. The real-time skills provided by the existing real-time middlewares used in robotics are not sufficient. To do this, we plan to either directly generate real-time tasks for the operating system, or to develop a light RT middleware that would allow to better control the real-time execution. Second, the pWCET estimation can also provide an estimate of the time taken by the *glue* (i.e. the middleware and some calls like reading/writing within ports), from the measurements that contain both codel calls and components triggers. This glue estimation is available but has not been taken into account in the WCRT analysis yet. Finally, we plan to enhance the probabilistic WCRT computation by taking into account preemptions and delays probabilities, in order to have even more precise schedulability results.

V. ACKNOWLEDGEMETS

This work is partially supported by the CPSELabs project funded by European Community's Horizon 2020 Programme under grant agreement no 644400.

REFERENCES

[1] A. Cervin, B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *RTCSA 2004*, Göteborg, Sweden, 2004.

[2] D. Henriksson, A. Cervin, J. Åkesson, and K.-E. Årzén, "On dynamic real-time scheduling of model predictive controllers," in *CDC 2002*, Las Vegas, NV, 2002.

[3] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time," in *ICRA 2014*, Hong Kong, China, 2014.

[4] M. Sanfourche, V. Vittori, and G. Lebesnerais, "Evo: A Realtime Embedded Stereo Odometry for MAV Applications," in *IROS 2013*, Tokyo, Japan, 2013.

[5] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive Neighborhood Selection for Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images," in *IROS 2012*, Vilamoura, Portugal, 2012.

[6] F. Steinbrücker, J. Sturm, and D. Cremers, "Volumetric 3D Mapping in Real-Time on a CPU," in *ICRA 2014*, Hong Kong, China, 2014.

[7] L. Wang, M. Liu, and M. Q.-H. Meng, "Hierarchical Auction-Based Mechanism for Real-Time Resource Retrieval in Cloud Mobile Robotic System," in *ICRA 2014*, Hong Kong, China, 2014.

[8] G. Biggs, K. Fujiwara, and K. Anada, "Modelling and Analysis of a Redundant Mobile Robot Architecture Using AADL," in *SIMPAR 2014*, Bergamo, Italy, 2014.

[9] C. Schlegel, A. Steck, D. Brugali, and A. Knoll, "Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering," in *SIMPAR 2010*, Darmstadt, Germany, 2010.

[10] F. Singhoff, J. Legrand, L. Nana, and L. Marc, "Cheddar : a Flexible Real Time Scheduling Framework," *ACM SIGAda Ada Letters*, vol. 24, no. 4, pp. 1–8, 2004.

[11] N. Gobillot, C. Lesire, and D. Doose, "A Modeling Framework for Software Architecture Specification and Validation," in *SIMPAR 2014*, Bergamo, Italy, 2014.

[12] N. Gobillot, D. Doose, C. Lesire, and L. Santinelli, "Periodic state-machine aware real-time analysis," in *ETFA*, Luxembourg, Luxembourg, 2015.

[13] P. Soetens and H. Bruyninckx, "Realtime Hybrid Task-Based Control for Robots and Machine Tools," in *ICRA 2005*, Barcelona, Spain, 2005.

[14] M. Desnoyers and M. Dagenais, "The LTTng tracer: A Low Impact Performance and Behavior Monitoring for GNU/Linux," in *Ottawa Linux Symposium*, Ottawa, Canada, 2006.

[15] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM TECS*, vol. 7, no. 3, 2008.

[16] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla, "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs," in *ECRTS 2012*, Pisa, Italy, 2012.

[17] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart, "On the Sustainability of the Extreme Value Theory for WCET Estimation," in *WCET 2014*, Madrid, Spain, 2014.

[18] F. Guet, L. Santinelli, and G. Morio, "On the reliability of the probabilistic worst-case execution time estimates," in *ERTS*, Toulouse, France, 2016.

[19] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour, *A practitioner's handbook for real-time analysis*. Kluwer Academic Publishers, 1993.

[20] S. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Systems*, vol. 24, no. 1, pp. 93–128, 2003.

[21] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph real-time task model," in *RTAS*, Chicago, IL, USA, 2011.

[22] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE T-RO*, vol. 23, no. 1, pp. 34–46, 2007.