An open-source architecture for simulation, execution and analysis of real-time robotics systems

Dennis Leroy Wigand, Pouya Mohammadi, Enrico Mingo Hoffman, Nikos Tsagarakis, Jochen Steil, Sebastian Wrede

An Open-Source Architecture for Simulation, Execution and Analysis of Real-Time Robotics Systems

Dennis Leroy Wigand[1], Pouya Mohammadi[2], Enrico Mingo Hoffman[3]
Nikos G. Tsagarakis[3], Jochen J. Steil[2], Sebastian Wrede[1]

Abstract— **The specification and analysis of the timing are an integral part of a robotics system that requires to be highly reliable. Especially since the demand for robots, which are applied in collaborative environments, is increasing drastically, robots need to be even more reliable and safe. In this paper, we propose a workflow for timing specification and analysis of real-time sensitive component-based robotics systems. Further, we introduce CoSiMA, a C++ based architecture that combines technologies, which are well-known in the domain of robotics. CoSiMA offers the ability to model, simulate, deploy, and analyze a robotics system on different robotic platforms. In addition to that, it offers a real-time safe mechanism to collect execution time data of a system, run in simulation or on the real hardware, to investigate and ensure the desired behavior of the robot. In order to depict the proposed workflow, we implemented an experimental system using CoSiMA, which lets the humanoid robot COMAN perform a *Zero Moment Point*-based walk on a straight line.**

I. INTRODUCTION

With the current rise of collaborative robots entering various fields of the industry, the specification and analysis of the execution time behavior (e.g., reaction times) becomes an even more crucial aspect in the development of robotics systems, especially considering safety and reliability [1]. In order to safely deploy these systems in environments, where a (timing) error could lead to a hazard, endangering the robot, the environment or even worse a human being, it is mandatory to provide guarantees on the system's behavior [2], [3]. While this is already necessary for non real-time applications, it is all the more mandatory for real-time component-based robotics systems (CBRS) [4]. Investigating the real-time execution behavior – although in other domains very common [5] – has not been a major concern in robotics, since recent research tends to address mainly functional capabilities [6], [7]. Some work even states that violating deadlines does not result in significant impact on their systems [2].

To the best of our knowledge, there are currently very few publications that point out that a timing specification and analysis needs to be an integral part of the development process, to identify and correct potential timing or performance problems early on, and reduce the development costs and person-hours for maintenance significantly [8], [9], [4]. We argue that it is not enough to integrate the timing aspect into the development process, but even more it is necessary to consider the separation of roles (SoR) in the different stages. Further, we claim that different roles have different non-functional requirements on the analysis and associated views.

Related work found in the literature of robotics that does consider timing as important mostly focuses only on determining the worst-case execution time (WCET) as well as the worst-case response time (WCRT) [2], [1], [3], [8], [9], [4], or on verifying the schedulability [10]. There are some publications that explain the importance of using the structural model information of a component-based system, to calculate end-to-end latencies [11], [6]. This alone, however, is not sufficient to clearly cover the timing behavior of a real-time CBRS. Firstly, there is a lack of specifications for robotics-specific non-functional requirements, such as the concept of *cause-effect chains* [6]. Secondly, in the current state-of-practice, the specification of the execution time behavior is mostly hidden in the component implementation [1], as well as in the hardware (interfaces), i.e., delays in the sensors and their communication [12]. Those timing details need to be made explicit. Domain-specific approaches exist in the literature that try to formalize the non-functional requirements for extensive model-based analyses and to facilitate the integration of existing analyzers and visualizations into the workflow via code-generation [1]. The majority of these requirements fail to cover essential robotics-specific aspects.

In order to address these problems, we propose a refined workflow for timing specification and analysis, separated along the involved roles and their associated concerns, to exploit the power of Separation of Concerns (SoC) and SoR to enhance the development process (Sec. II). Furthermore, we propose robotic-specific specifications that we deem necessary for our compliant human-robot interaction use cases. We introduce our CoSiMA architecture[1] (Sec. III) that enables the simulation of real-time constrained robotics systems and the analysis of specified timing requirements (Sec. IV). In the last part of this paper, we apply the introduced workflow to a humanoid walking control system in CoSiMA, and present situations in which the developer benefits from the analysis to prevent a faulty behavior (Sec. V).

[1]Technical Faculty, Bielefeld University, Germany, {dwigand, swrede}@techfak.uni-bielefeld.de
[2]Research Institute for Robotics and Process Control, TU Braunschweig, Germany, {p.mohammadi, j.steil}@tu-bs.de
[3]Department of Advanced Robotics, Istituto Italiano di Tecnologia, Genova, Italy {enrico.mingo, nikos.tsagarakis}@iit.it

[1]http://cogimon.github.io

II. Timing Specification and Analysis

When designing a real-time-constrained CBRS that strongly relies on data-flow communication, it is not sufficient to solely focus on the WCET, WCRT, or schedulability of the components and system respectively, even though those are very important aspects. In order to ensure that a system is behaving as intended by the developer, two aspects need to be considered: Firstly (II-A), to avoid overwhelming the developers with various different tasks, the concepts of SoC and SoR can be employed, since we argue that each role has different functional requirements on the timing specification and analysis, as well as on the visualizations. This splits up the tasks, relieving the developers and offering the possibility for a modular workflow. Secondly (II-B), the specification needs to be enriched by non-functional robotics-specific requirements w.r.t. the intended behavior of a robot.

A. Separation of Roles and Concerns

For the Separation of Roles, we draw on the roles described by the RobMoSys[2] project, in particular the *Component Supplier*, the *System Builder*, and the *Performance Designer*.

1) Component Supplier (CS): The Component Supplier, aka Component Developer, offers computational units in form of software components that provide or require specific interfaces. A component, for instance, could be a low-level PID controller, a high-level planning component, etc. This role is naturally interested in determining general execution properties, such as the WCET and the memory consumption. Furthermore, this role particularly requires the ability to investigate the variance of execution times as well as the violations of deadlines. Apart from investigating these concerns, the Component Supplier also needs to explicitly specify the (desired) execution properties (i.e., WCET). This specification is then used as ground-truth for later validation and as base information for other roles. Suitable visualizations for the concerns of this role found in the literature are histograms or classical one-dimensional plots.

2) System Builder (SB): The main task of the System Builder, aka System Integrator, is to realize the system architecture designed by the Application-Domain Expert (ADE), by drawing on the software components provided by the CS. The SB requires to investigate the WCRT and End-2-End latency of a (sub-)set of components respectively. Together with the ADE, component overarching constraints need to be specified. Depending on the desired application, different precedence constraints [13] have to be defined (in form of so called Precedence Task Graphs (PTG)) as a precondition for the execution of a component and to structure the trace of data that will be processed by a component. This way it is specified which parts of the system can be executed in parallel and which components need to wait for others to finish, e.g., because they depend on previously processed results from another component.

[2]`https://robmosys.eu/wiki/general_principles:ecosystem:roles`

3) Performance Designer (PD): The Performance Designer takes care of realizing the system planned by the SB (including the components provided by the CS) in terms of activities that are assigned to cores. Here, the challenge is not only to make the system schedulable, but also to meet the different requirements, specified by the other roles. Therefore, this role has very different concerns and thus requires other support as compared to the previously mentioned roles. Basically, each component needs to be assigned to an activity, which in turn executes the containing components sequentially. Every activity is defined by its activation, which is either periodic or aperiodic, and by its priority, and optional, particular scheduler. Activities are executed in parallel and can be assigned to different cores. Since the mapping from the requirements to the actual schedulable system is a non-trivial task, the PD needs to be able to investigate various aspects: The specified PTG needs to be compared to the actual execution and data-flow of the system. Hence, a timing diagram with data-flow information is suitable (see Fig. 9).

To solely verify the execution order a classical timing diagram seems to be sufficient. Further, the PD needs to ensure that the specific parts of the PTG are actually schedulable, and check if the mapping from the graph to activities can be optimized. Otherwise, the development process needs another iteration, triggering the SB to optimize the PTG to hopefully enable the PD to find a schedulable solution. In addition to that the core utilization as well as the distribution of activities on cores and components on activities needs to be investigated.

In addition to that, activation constraints [6] (i.e., activation frequency/period) and the activation source [6] (e.g., event- or timer-triggered) need to be explicitly specified.

B. Robotics-Specific Requirements

Each role has to care for different concerns and needs to specify their non-functional properties, which are then used by the other roles. However, the requirements mentioned in Sec. II-A are not sufficient to cover the timing of a CBRS completely. In the following, several robotics-specific requirements are added and motivated per role.

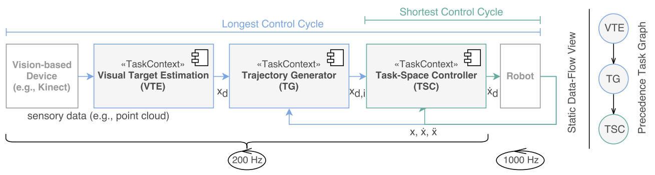

Shortest Control Cycle
Longest Control Cycle
Vision-based Device (e.g., Kinect)
«TaskContext» Visual Target Estimation (VTE)
«TaskContext» Trajectory Generator (TG)
«TaskContext» Task-Space Controller (TSC)
Robot
x_d
$x_{d,i}$
$\dot{x}_d$
sensory data (e.g., point cloud)
$x, \dot{x}, \ddot{x}$
200 Hz
1000 Hz
Static Data-Flow View
VTE
TG
TSC
Precedence Task Graph

Fig. 1. Static view of a simple closed-loop velocity control system. The desired point in the task-space is detected and processed by a vision pipeline (VTE) and afterwards send to an online trajectory generator (TG), which sends the single points on the trajectory to a task-space velocity controller (TSC) that ultimately controls the robot with velocity commands. While in this example the vision pipeline is able to produce a new trajectory every 5ms, the robot needs to receive a new velocity command every millisecond to ensure the desired behavior. Thus, the associated precedence graph of the components, is not sufficient to capture the systems behavior including the timing aspects (cf. II-B.1).

1) System Builder: In addition to the concerns discussed in Sec. II-A.2, the pure specification of PTGs are not sufficient to cover the needs of robotic systems, since there are cases in which the specification of the data-flow needs to be decoupled from the execution order. For instance, the dataflow of the system shown in Fig. 1 can easily be described as a PTG. However, the execution order is not modeled correctly, since the components need to run with different frequencies rather than in this exact sequence, to ensure a stable control behavior of the robot. A direct consequence is that components may process old input data, which is an aspect that frequently occurs in robotics. Hence, we argue that the System Builder requires to be able to specify constraints on the data, such as how far an input can date back in order to be still valid to be processed (e.g., output constraints). Another important point is the definition of control loops or *cause-effect chains* as proposed by [6]. In our case, those are sequences of components that begin with sensor data and end with a resulting actuator command. These chains need to be unambiguously classified into *trigger-* and *data-chains*, since this has an impact on the End-2-End latency analysis [11] and helps to specify the desired behavior more clearly. While trigger-chains can realize the behavior shown in Fig. 1, data-chains are more similar to a PTG, where a sequence of components is executed sequentially by a data-event. As an additional requirement for the PD, the SB (together with the ADE) is able to specify the maximum time/minimal frequency that the shortest/longest control cycle is allowed to take/to be updated, in order to keep the robot stable. A complete control cycle begins with a feedback data of the robot and ends with the produced command that is based on the feedback data and send to the robot. Similarly, a short control cycle could also encompass a single controller sending commands to the robot (as long as a stable behavior is imposed).

2) Performance Designer: Apart from the general concerns, the Performance Designer needs to also consider the robotics-specific aspects specified by the System Builder. This means not only the additional investigation of the End-2-End latency of the longest and shortest control cycle, but also to incorporate this new information in the mapping of components to activities and cores. While these new requirements seen individually do not seem to be overly expensive, the entire set of requirements however, yields a highly complex optimization problem that requires different concern-specific views in terms of analyses and visualizations to support the role of the Performance Designer.

III. CoSiMA Architecture

Due to projects such as CogIMon[3], which focus on compliant interaction between robots and the environment, including humans, we need to be able to develop and simulate real-time-constrained CBRS, before deploying them on real robotic platforms. With the major aspect of safety and reliability in mind, we created CoSiMA: an architecture that is specially designed to support the component-based development of robotics systems, the analysis of the simulated systems in terms of behavior and timing, as well as the execution on real robotic platforms, including fully transparent switching with minimal development effort. CoSiMA incorporates different well-established technologies (R.1), such as OROCOS [14], ROS [15], RSB [16], and Gazebo [17], together with custom extensions to meet the requirements of

- R.1 providing a low entry barrier and not re-inventing the wheel.
- R.2 being as most compatible to existing software frameworks as possible.
- R.3 providing a framework with the necessary tools, interfaces and pre-designed components for CBRS.
- R.4 supporting the execution of real-time critical systems.
- R.5 keeping the implemented code agnostic to execution in simulation or on the real robot.
- R.6 maximizing the portability of controllers between different kinds of robotic platforms.
- R.7 being able to introspect the (timing) behavior of a system.

CoSiMA is divided into three major parts: execution, simulation, and modeling environment. In the course of this paper, we will only cover the first two parts. For information on the modeling part, please refer to [18].

The Orocos Real-Time Toolkit (RTT) [14] aka OROCOS resembles a C++ framework, which allows the implementation of real-time and non-real-time control systems in a component-based way. OROCOS was chosen as the execution environment for CoSiMA because of two reasons: Firstly (R.3), for offering a component-based approach for the implementation of functional capabilities, which has proven to be a widely adopted and effective solution [19]. Secondly (R.4), for the support of real-time control systems and the integration with real-time frameworks, such as Xenomai[4]. In the context of this paper, we refer to a robotics system as a real-time system, which requires a response to an event precisely within a bounded time [20].

As simulation environment the well-established and open-source physics simulator Gazebo [17] was chosen. Gazebo is maintained by the Open Source Robotics Foundation[5] and is one of the most common simulators used in the field of robotics [21], since it also integrates smoothly with ROS [22] (R.2).

A. The Architecture

CoSiMA supports both, non-real-time as well as real-time sensitive implementations, but deploys them separately. As it can be seen in Fig. 2, everything is deployed in a global execution environment, e.g., a workstation for instance. Inside that global environment, real-time-constrained components are deployed in an OROCOS environment,

[3]Cognitive Interaction in Motion `https://www.cogimon.eu`

[4]`https://www.xenomai.org`

[5]`https://www.osrfoundation.org`

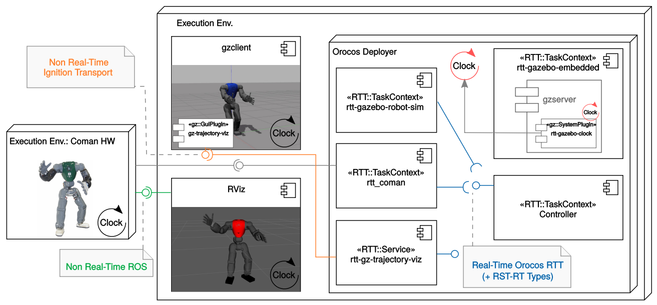


Fig. 2. The CoSiMA deployment structure consists of real-time as well as non-real-time environments. Using specific data types and transports, communications between these environments can be established. Further, CoSiMA allows transparent switching between simulation and real hardware.

whereas components such as the RViz visualization lives in the global environment itself. In case of simulating a system, the Gazebo Client, which performs the rendering of the simulation, is deployed without real-time constraints, while the Gazebo Server is instantiated as an OROCOS component[6] and thus lives inside the OROCOS environment, which constitutes two advantages: First, other OROCOS components in the same environment can directly access the Gazebo API, since everything runs in the same process. Second, using a plugin[7], Gazebo is able to propagate the simulated time into the OROCOS environment. This way, the simulation and the control components are synced time-wise.

B. Communication and Semantic Data Types

Inter and intra communication of the real-time and non-real-time environments is currently supported via data types provided by ROS and RSB/RST. In case of inter environment communication, which here basically means inter process communication, a special plugin for OROCOS is needed that ensures real-time safe message exchange for a particular transport. Since OROCOS is also shipped with ROS, an integration containing such a plugin[8] is directly provided by ROS. In order to keep CoSiMA ROS-agnostic, a more lightweight transport is provided by RSB and the associated protocol-buffer based types collected in RST(RT)[9]. RST(RT) contains real-time safe data types that add robotics-specific semantics to the communication. All the necessary plugins are available at `https://www.github.com/corlab`.

C. Integration of Robotic Interfaces (R.5, R.6)

CoSiMA has been designed to be able to simulate and control any robot of any complexity: from simple manipulators to humanoid robots. The key component which permits to interface with the simulated robot is called `rtt-gazebo-robot-sim`[10]. This component, starting from the URDF and SRDF files, which describes the kinematics, dynamics, chains, controllers, and sensors arrangement, automatically generates a set of ports which can be used to send commands as well as read sensor data to and from the simulation. Leveraging the concept of kinematic chains, we are able to design a robot control interface that is determined by the different kinematic chains, consisting of a sequence of joints with available control modes. Further, the component provides a set of operations which are used to setup other components that need to connect with it, providing information used to automatically connect the necessary ports (e.g., to receive the robot's feedback). This automatically determines the data type (see III-B) necessary for communication as well as the dimension of the expected command. Finally, the component (and hence the ports) are parametrized over the name of the robot making it possible to instantiate and control more robots of different or the same kind.

For the real hardware, specular components, which open the same ports and offer the same operations, are provided. The currently supported robots include the KUKA LWR 4+ as well as the COMAN. For this work we present the interface component for the COMAN: `rtt_coman`[11]. It connects via a driver to the *ethernet*-based robot [23]. The controllers do not know if they are connected to the real robot or the simulated one (R.5). This way, moving from simulated to the real hardware is simple and straightforward solely by exchanging the robot interface component in the configuration of the system (i.e., in the OROCOS Program Script file).

We recall that the kinematic chains, defined by the configuration from the URDF, SRDF, and control mode, determine the interface of the "to be controlled"-robot. Based on the kinematic chains, it can be determined whether a controller is suitable to control a specific robot. This is the case, only if the interface description of such a controller matches the kinematic chain's interface R.6. However, whether it makes sense to use a controller designed for robot A on robot B, is still subject to the System Builder's opinion.

IV. Timing Introspection Extension

In order to be sure that a real-time constrained robotics system behaves according to the developer's intentions, a mechanism is needed to introspect the system's execution while also collecting data for further analysis and verification of the specified timing requirements. For most of the developers that use OROCOS or a similar framework, the execution of their components (tasks) is inaccessible and thus not investigatable. Since we are doing physical human/environment-robot interaction, we need to be certain that our systems execute and behave as intended. Otherwise, serious issues might arise that effectively damage the environment, the robot, or even worse hurt a human.

[6]`https://github.com/corlab/rtt-gazebo-embedded` forked from Institut des systmes intelligents et de robotique.

[7]`https://github.com/corlab/rtt-gazebo-clock-plugin` extracted and improved from Johns Hopkins University LCSR.

[8]`http://wiki.ros.org/rtt_ros_integration`

[9]`http://docs.cor-lab.org/rst-manual/trunk/html/index.html`

[10]`https://github.com/cogimon/rtt-gazebo-robot-sim`

[11]`https://gitlab.advrcloud.iit.it/advr_humanoids/rtt_coman/tree/master`

A. OROCOS Task Execution

In OROCOS, executing a component basically means triggering a component's internal `Execution Engine`, which in turn e.g., handles incoming data and ultimately calls the `updateHook()` of a component. The updateHook() represents the function that contains the e.g., calculations for the (next) control signal that is going to be send to the robot. To trigger a component's Execution Engine, different kinds of `Activities` can be used. The type of an Activity is defined by its *periodicity* and *priority*, the *core* it should be executed on, and whether it should use a real-time or non-real-time *scheduler*. Note that one Activity can only trigger one associated Execution Engine. Activities however are triggered by `Threads`. There are also different kinds of Threads, which mostly fit a specific type of Activity. In contrast to Activities, Threads can trigger multiple Activities sequentially in the order they are started. For more detailed information, please refer to [14]. In the following we will use the term Activity as a synonym for Thread and the execution of a component as a synonym for triggering an Execution Engine.

B. CoSiMA's Introspection Architecture (R.7)

There are different approaches to gain insights into the execution of a system: ranging from hardware platform emulators [10], over just in time recompilation of x86 machine code for accurate simulation [24], to sampling-based approaches [20]. Since modeling the entire real-time environment, necessary for emulation, is not trivial [10], and any overhead that slows down the execution significantly needs to be avoided as much as possible, a sampling-based approach was chosen. This, however, trades accuracy for performance, enabling the introspection to work during real runs of a system.

In light of further requirements, such as sampling with the same synchronized (simulation) time that our OROCOS components are executed with, and being able to easily process (e.g., live stream) our data via a middleware of our choice, without having I/O operations that break the real-time constraints, we choose a "native" OROCOS implementation.

1) Introspection Mechanism: The introspection mechanism consists of two parts (see 3): (1) An interface that allows to turn the introspection for specific ports and lifecycle states of a component "on" and "off", as well as to specify the capacity of the internal storage for the samples. This interface specializes the OROCOS `RTT::TaskContext` and acts as base-class, which components need to inherit from to become introspectable. The mechanism covers the instant that data is send and received through a port, as well as the tracking of begin and end of the lifecycle states. The implementation itself is real-time safe and tested on Xenomai. (2) A real-time safe logging component `IntrospectionReporter` will then collect all the samples from the introspectable components and either writes them into a file, or sends them out using a specific transport, depending on the marshallers that are used.

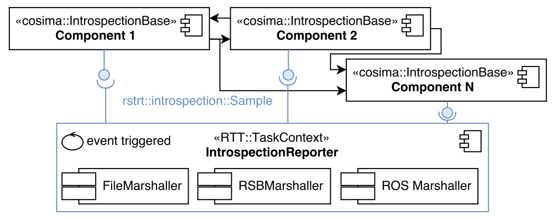


Fig. 3. A system containing three introspectable components that are registered with the Introspection Reporter component, which collects the execution samples and publishes them via the available marshallers.

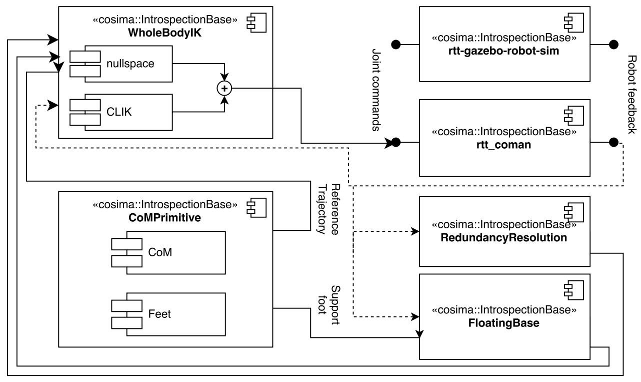


Fig. 4. Component architecture of the COMAN experiment.

2) Visualization and Analysis: Furthermore, CoSiMA comes with a light-weight visualization and analysis tool (see Fig. 9), which is inspired by the work of [25]. It offers an enhanced timing diagram enriched with data-flow information, which allows the analysis of the shortest and longest control cycle as well as the detection of misalignments related to the frequency or starting order of components. Apart from that, it allows the introspection of the distribution of components inside activities, to investigate deadline violations and potential for reorganization.

V. Case Study

In this section, the workflow presented in Sec. II is applied using CoSiMA to a timing-sensitive use-case involving the humanoid robot COMAN (see Fig. 5), which was developed at the Italian Institute of Technology (IIT) during the AMARSi Project[12]. The structure of the section first presents the use-case and then discusses the workflow along the introduced roles analogous to Sec. II.

A. COMAN experiment

In this experiment, the COMAN should perform a straight walking motion. To achieve this, a system comprising the following five components is used:

FloatingBase (*base*) is responsible to compute the position and orientation of the robot's floating base. Its inputs are the current robot configuration as well as the active support foot.

CoMPrimitive (*com*) provides reference trajectories for the center of mass (CoM) and feet (swing/support) of the

[12]`http://www.amarsi-project.eu`

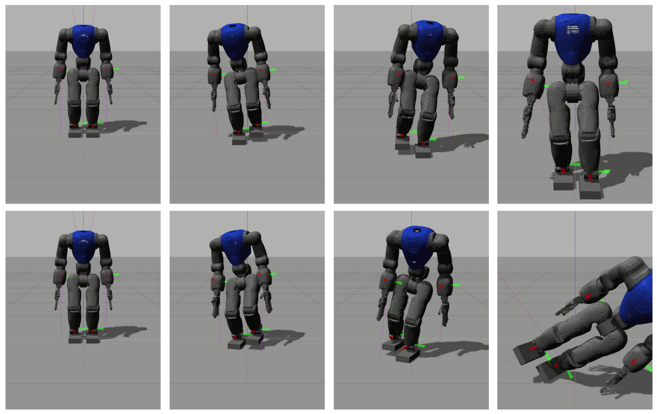


Fig. 5. Walking experiments with the COMAN robot. The top row demonstrate the case when all components meet their timing requirements according to the specification. In the second row, CoMPrimitive has an intentionally slower activation frequency, which leads to a faulty behavior.

robot to the inverse kinematics component (WholeBodyIK). For this test, the robot should perform a walk on a straight line. The other output of this component is an indicator for the current swing foot (left/right) which is needed by the FloatingBase component.

WholeBodyIK (*ik*) solves the inverse kinematics of the humanoid robot. Given a desired Cartesian velocity, current body configuration, null-space choice, and pose of the floating base, it solves the IK using a closed loop inverse kinematic (CLIK) method and sends the results to the robot.

RedundancyResolution (*rr*) provides a null-space joint motion behaviour that allows the WholeBodyIK component to use different solutions among infinite possible ones. For this particular case, we try to find joint values that keep the robot's torso near its upright configuration.

rtt-gazebo-robot-sim (*robot_gazebo*) resembles the interface to the robot. It provides the robot's feedback in terms of joint position, velocity and torques to the components and forwards control commands to the robot or – in case of Fig. 5 – the simulation model in Gazebo.

Fig. 4 depicts the interconnections between these components. The most important and computationally demanding component is WholeBodyIK. It relies on the feedback from the robot, as well as the input from all other components. An example of a timing failure is when the reference trajectories from CoMPrimitive are delayed and not match the feedback from the robot anymore. In that case, there is an abnormally large error between the desired and real values for different poses of the robot. This is depicted in the bottom row of Fig. 5, where the robot diverges from its designated path immediately after initiating a *Zero Moment Point*-based [26] walk on a straight line. Another case, which is excluded in this setup in order to keep the notations light, is when there is a stabilizer, which corrects the desired CoM trajectories. In case of an unexpected delay, the stabilizer would attempt to fix an "old" trajectory that was executed by the robot.

B. Workflow: Component Supplier

As described in Sec. II-A.1, the Component Supplier provides the necessary functional capabilities in form of components and associated execution properties. For this case study, we focus on the individual WCET (see Fig. 6), which were approximated through multiple runs. More accurate results can be achieved by using methods such as [27]. However, since the focus here is on the workflow and specification, it is sufficient to take the maximum execution time of multiple runs. Fig. 8 shows a JSON[13]-based specification for the FloatingBasePose component.

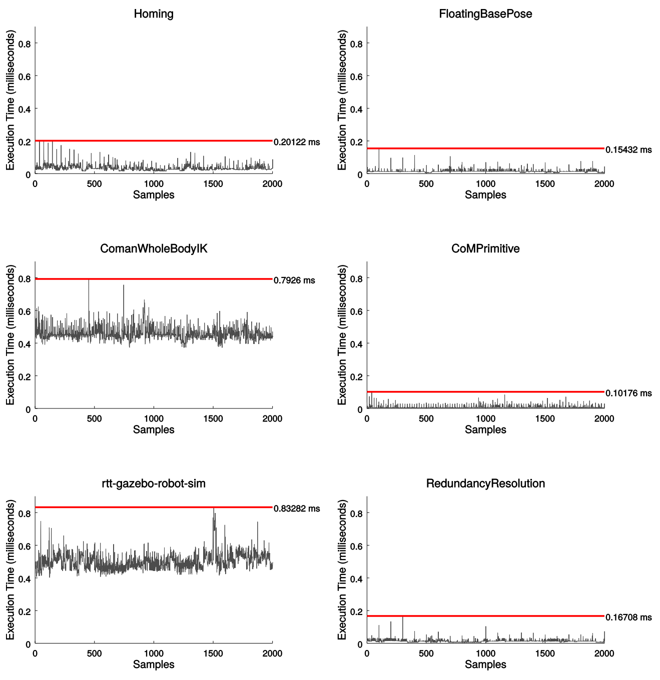


Fig. 6. Sample-based execution time of the involved components of the COMAN experiment. The approximated WCET is indicated by the red line.

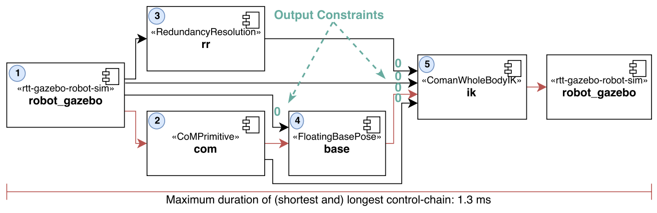
C. Workflow: System Builder

The System Builder enriches the specification from the Component Supplier with application-specific information: As introduced in Sec. II-A.2, constraints on the data are added in form of output constraints (see Fig. 8 (orange part)) that determine if received data is valid to be processed based on the cycle (time) it was received. These constraints can also be seen in the precedence task graph (PTG) in Fig. 7 (green numbers). The PTG is realized as an annotated DOT[14] graph, which includes the starting order as well. Here, we see that the ComanWholeBodyIK instance *ik* demands all its input data to be from the same and current cycle, whereas the FloatingBasePose instance *base* only constraints one of its inputs to be from the current cycle, namely the feedback from the robot.

To further add the robotics-specific aspects (see Sec. II-B.1) to the specification, generally the longest and shortest

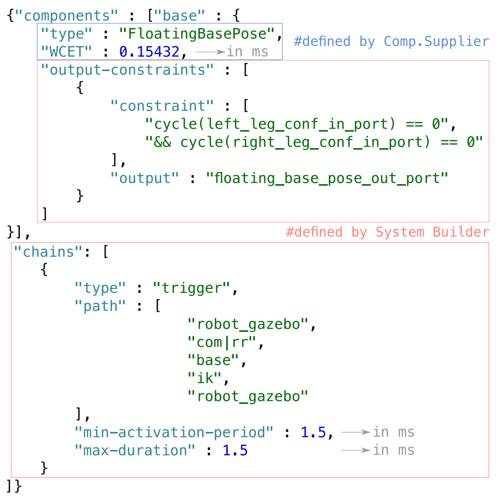
[13]`http://json.org`
[14]`http://www.graphviz.org/doc/info/lang.html`

cause-effect chain should be described. However, in this example the longest is also the shortest chain, since there is no shorter one than the chain, depicted in Fig. 7, which would be able to impose a stable behavior on the robot.



Output Constraints
3 «RedundancyResolution» rr
1 «rtt-gazebo-robot-sim» robot_gazebo
2 «CoMPrimitive» com
4 «FloatingBasePose» base
5 «ComanWholeBodyIK» ik
«rtt-gazebo-robot-sim» robot_gazebo
0
0
0
0
0
Maximum duration of (shortest and) longest control-chain: 1.3 ms

Fig. 7. Data-flow specification in form of a PTG, including the starting order of the components (blue circle), the critical path (red arrows), and data constraints (0 $\widehat{=}$ data from current cycle; negative numbers $\widehat{=}$ previous cycles).



```
{"components" : ["base" : {
    "type" : "FloatingBasePose",   #defined by Comp.Supplier
    "WCET" : 0.15432,  ⟶in ms
    "output-constraints" : [
        {
            "constraint" : [
                "cycle(left_leg_conf_in_port) == 0",
                "&& cycle(right_leg_conf_in_port) == 0"
            ],
            "output" : "floating_base_pose_out_port"
        }
    ]
}],                                        #defined by System Builder
"chains": [
    {
        "type" : "trigger",
        "path" : [
                    "robot_gazebo",
                    "com|rr",
                    "base",
                    "ik",
                    "robot_gazebo"
        ],
        "min-activation-period" : 1.5,  ⟶in ms
        "max-duration" : 1.5            ⟶in ms
    }
]}
```

Fig. 8. Excerpt of the specification for the COMAN experiment, depicting the definition of the WCET, the data constraints of a component, as well as the control chain.

D. Workflow: Performance Designer

The following set of periodic activities is chosen to meet the requirements defined by the timing specification.

Activity 1 (period 1.0ms, core 1) *robot_gazebo*
Activity 2 (period 1.3ms, core 0) *com*, *base*, and *ik*
Activity 3 (period 1.3ms, core 0) *rr*
Activity 4 (period 0.0ms, core 2) IntrospectionReporter *ir*

The rationale to combine *com*, *base*, and *ik* in one activity, is based on the data constraints of *ik*, which demand the synchronicity of the received data. Using the activity to represent a data-chain between the three components ensures a valid data-flow (see Fig. 9). The robot interface is assigned to an empty core to avoid preemption and to ensure stable communication with the robot. Although the RedundancyResolution component is a part of the control system of the robot, it resembles a secondary task with lower priority. Therefore, it does not lead to a huge error, if the component is misaligned and sends data based on a previous cycle. To not add any kind of delay, it runs in a separate activity in parallel to the main control activity (2). The activity containing the introspection component is an exception. It is not periodically updated, but event-triggered. In this case, it activates on every incoming introspection sample. Apart from that, it needs to be verified that the End-2-End Latency (1.3 ms) is not greater than the specification (1.5 ms) allows (see Fig. 9). The resulting behavior is displayed in Fig. 5.

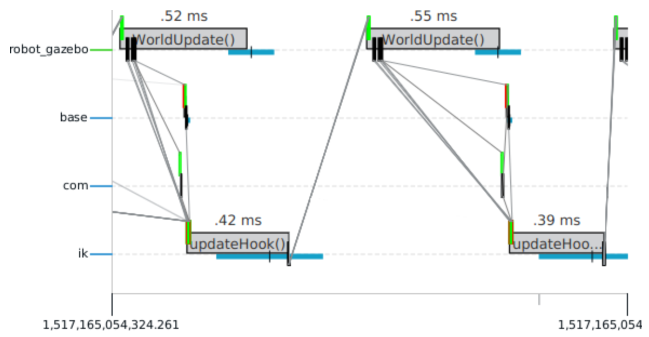


Fig. 9. Time diagram with data-flow information of the correct specification, which results in the behavior of Fig. 5 (top row). For the sake of readability, the RedundancyResolution component was neglected. *com*, *base*, and *ik* are deployed in the same activity (indicated by the colored line next to the names). As it can be seen, the components are executed in the right order and *ik* only receives input data that is based on the current cycle. Short vertical lines represent output (black) and input ports with (green $\widehat{=}$ new, cyan $\widehat{=}$ old, red $\widehat{=}$ no) data.

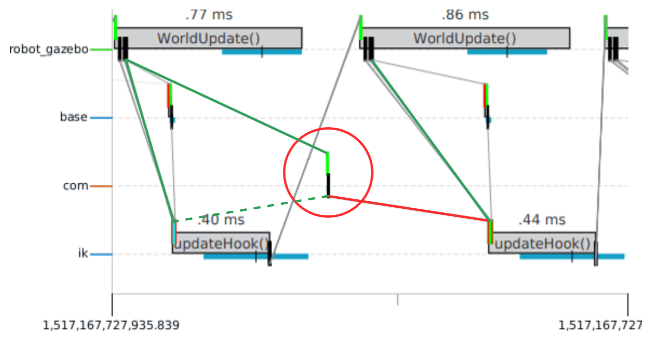


Fig. 10. This diagram shows a wrong specification, in which *com* is running in a separate activity and is slightly misaligned to *base* and *ik*. Here, *com* is executed after *ik*, so it sends its data to the next iteration of *ik*. This means that *ik* operates on "old" data, which causes the behavior seen in Fig. 5 (bottom row). The green lines indicate the correct data-flow, whereas the red lines indicate the actual but wrong data transfer.

E. Consequences of Neglecting Robotics-Specific Aspects

In the case the robotics-specific timing aspects are neglected for this experiment, one activity could be used containing a sequence of all involved components. This, however, would corrupt the robot's behavior, since the minimal update frequency to keep the robot stable cannot be met. In contrast to that, without the constraints on valid data, the Performance Designer would have no clue, which component can be paralleled and how, in order to ensure the desired behavior. Fig. 10 displays the case where *com* runs slightly misaligned in parallel to the sequence of *base* and *ik*. It can be clearly seen that the *ik* component is operating on data from *com*, which is based on a previous cycle. The resulting faulty behavior is expressed by the bottom row of Fig. 5.

VI. Conclusion

This paper proposed an extension to the development process of component-based systems in the domain of robotics, which focuses on the integration of robotics-specific timing aspects as well as the Separation of Roles and Concerns. This offers the developers a way to define more fine-grained requirements on real-time sensitive systems, which can be evaluated in preface or in simulation, and thus, increases the reliability and safety. To not only specify but to also create, simulate and evaluate a component-based robotics system, we introduced CoSiMA, which is based on well-known technologies and offers transparent development as well as support for interchangeability between real-time controller systems and robotic platforms. The presented development process was evaluated using an experimental system with the humanoid robot COMAN.

As future work, we will target the integration of the specification requirements into the modeling environment of CoSiMA, which was published in [18]. This will include a model-driven formalization in form of a domain-specific language, to increase the readability and to enable model-based verification. Further, we would like to evaluate the suitability of LTTng [20] as alternative to the current sampling mechanism.

ACKNOWLEDGMENT

This work was supported by the European Communitys Horizon 2020 robotics program ICT-23-2014 under grant agreement 644727 - CogIMon and was supported by the Cluster of Excellence EXC-277: Cognitive Interaction Technology (CITEC) at Bielefeld University.

References

[1] A. Lotz, A. Hamann, R. Lange, C. Heinzemann, J. Staschulat, V. Kesel, D. Stampfer, M. Lutz, and C. Schlegel, “Combining robotics component-based model-driven development with a model-based performance analysis,” in *IEEE Int. Conf. Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Dec 2016, pp. 170–176.

[2] N. Gobillot, F. Guet, D. Doose, C. Grand, C. Lesire, and L. Santinelli, “Measurement-based real-time analysis of robotic software architectures,” in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 3306–3311.

[3] L. K. Chong, C. Ballabriga, V. T. Pham, S. Chattopadhyay, and A. Roychoudhury, “Integrated timing analysis of application and operating systems code,” in *IEEE Real-Time Systems Symp. (RSS)*, Dec 2013, pp. 128–139.

[4] J. Carlson, “Timing analysis of component-based embedded systems,” in *ACM SIGSOFT Symp. Component Based Software Engineering*. ACM, 2012, pp. 151–156.

[5] K. Arnold, *Embedded Controller Hardware Design*. L L H Technology Publishing, 2000.

[6] A. Lotz, A. Hamann, I. Lütkebohle, D. Stampfer, M. Lutz, and C. Schlegel, “Modeling non-functional application domain constraints for component-based robotics software systems,” *arXiv preprint arXiv:1601.02379*, 2016.

[7] L. Muratore, A. Laurenzi, E. Mingo Hoffman, A. Rocchi, D. G. Caldwell, and N. G. Tsagarakis, “On the design and evaluation of xbotcore, a cross-robot real-time software framework,” *J. of Software Engineering for Robotics (JOSER)*, vol. 8, pp. 164–170, 2017.

[8] J. Kraft, “Enabling timing analysis of complex embedded software systems,” Ph.D. dissertation, Mälardalen University Press, Aug. 2010. [Online]. Available: http://www.es.mdh.se/pdf_publications/1912.pdf

[9] M. Bohlin, Y. Lu, J. Kraft, P. Kreuger, and T. Nolte, “Simulation-based timing analysis of complex real-time systems,” in *IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications*, Aug 2009, pp. 321–328.

[10] D. Decotigny and I. Puaut, “Artisst: an extensible and modular simulation tool for real-time systems,” in *IEEE Int. Symp. Object-Oriented Real-Time Distributed Computing (ISIRC)*, 2002, pp. 365–372.

[11] S. Mubeen, J. Mäki-Turja, and M. Sjödin, “Translating end-to-end timing requirements to timing analysis model in component-based distributed real-time systems,” *ACM SIGBED Review*, vol. 9, no. 4, pp. 17–20, 2012.

[12] M. Neunert, T. Boaventura, and J. Buchli, *Why Off-The-Shelf Physics Simulators Fail In Evaluating Feedback Controller Performance - A Case Study For Quadrupedal Robots*. World Scientific, 2016, pp. 464–472. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/9789813149137_0055

[13] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Sci. & Business Media, 2011, vol. 24.

[14] P. Soetens, “A software framework for real-time and distributed robot and machine control,” Ph.D. dissertation, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium, May 2006, http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf.

[15] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” 2009.

[16] J. Wienke and S. Wrede, “A middleware for collaborative research in experimental robotics,” in *IEEE/SICE Int. Symp. System Integration (SII)*, Dec 2011, pp. 1183–1190.

[17] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[18] D. L. Wigand, A. Nordmann, N. Dehio, M. Mistry, and S. Wrede, “Domain-Specific Language Modularization Scheme Applied to a Multi-Arm Robotics Use-Case,” *Journal of Software Engineering for Robotics (JOSER)*, vol. 8, no. 1, pp. 45–64, 2017.

[19] M. Y. Jung and P. Kazanzides, “An architectural approach to safety of component-based robotic systems,” in *IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2016, pp. 3360–3366.

[20] F. Rajotte and M. R. Dagenais, “Real-time linux analysis using low-impact tracer,” *Advances in Computer Engineering*, vol. 2014, 2014.

[21] E. Mingo Hoffman, S. Traversaro, A. Rocchi, M. Ferrati, A. Settimi, F. Romano, L. Natale, A. Bicchi, F. Nori, and N. G. Tsagarakis, “Yarp based plugins for gazebo simulator,” in *Modelling and Simulation for Autonomous Systems*, J. Hodicky, Ed. Cham: Springer Int. Publishing, 2014, pp. 333–346.

[22] M. Ferrati, A. Settimi, and L. Pallottino, “Ascari: A component based simulator for distributed mobile robot systems,” in *Modelling and Simulation for Autonomous Systems*, J. Hodicky, Ed. Cham: Springer Int. Publishing, 2014, pp. 152–163.

[23] E. Mingo Hoffman, N. Perrin, N. G. Tsagarakis, and D. G. Caldwell, “Upper limb compliant strategy exploiting external physical constraints for humanoid fall avoidance,” in *IEEE/RAS Int. Conf. Humanoid Robotics (Humanoids)*, 2013, pp. 397–402.

[24] N. Nethercote and J. Seward, “Valgrind: a framework for heavyweight dynamic binary instrumentation,” in *ACM Sigplan notices*, vol. 42, no. 6. ACM, 2007, pp. 89–100.

[25] J. C. de Kergommeaux and B. de Oliveira Stein, “Pajé: An extensible environment for visualizing multi-threaded programs executions,” in *Euro-Par 2000 Parallel Processing*, A. Bode, T. Ludwig, W. Karl, and R. Wismüller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 133–140.

[26] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, “Biped walking pattern generation by using preview control of zero-moment point,” in *Int. Conf. Robotics and Automation (ICRA)*, vol. 2. IEEE, 2003, pp. 1620–1626.

[27] G. Bernat, A. Colin, and S. M. Petters, “Wcet analysis of probabilistic hard real-time systems,” in *Real-Time Systems Symp. (RSS)*. IEEE, 2002, pp. 279–288.