Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

EVALUATING AND OPTIMIZING COMPONENT-BASED ROBOT ARCHITECTURES USING NETWORK SIMULATION

Daniel Krauß
Fabian Paus
Nikolaus Vahrenkamp
Tamim Asfour

Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology
Kaiserstraße 12
76131 Karlsruhe, GERMANY

Philipp Andelfinger

TUMCREATE Ltd and
Nanyang Technological University
1 Create Way
Singapore 138602, SINGAPORE

ABSTRACT

Modern service and humanoid robots are comprised of multiple computers distributed among the robots' hardware. During task execution, several software components are executed in parallel on the connected machines. Due to the complex control loops and communication requirements of robot tasks, a suitable assignment of software components to the available hardware units is necessary to achieve low reaction times. Currently, there is a lack of works on approaches to evaluate intra-robot communication. We propose a coupling between the robotics framework ArmarX and the network simulator OMNeT++ to support the evaluation and optimization of robot architectures. Our approach allows unmodified robot components to communicate across simulated network interconnects. In a case study, we examine the influence of different hardware assignments of software components on task execution times. We show that the timing information present in the simulation-based evaluations enables more efficient hardware assignments when compared to static graph partitioning.

1 INTRODUCTION

Most modern robots are distributed systems comprised of specialized components. Robotics frameworks used to develop robot applications rely on a modular architecture that incurs computation and communication overhead compared to monolithic architectures. Since many robots execute precise or even safety-critical tasks, the runtime behavior of robotics systems is an important topic, which has, however, received only limited attention in the literature.

A variety of robotics frameworks are developed by different research groups (Elkady and Sobh 2012), the most well-established being the Robot Operating System (ROS) (Quigley et al. 2009), which is used both in research and industry (Guizzo and Ackerman 2012). Other examples include Yet Another Robot Platform (YARP) (Metta et al. 2006) and Orocos (Bruyninckx 2001). In our work, we examine the ArmarX framework (Vahrenkamp et al. 2015). Robotics frameworks provide the ability to create software modules and contain pre-built modules with specific robot functionalities such as image recognition or motor control. At the core of these frameworks, communication middlewares provide functionalities for the components to transparently communicate with each other over an underlying network topology. Communication between components is typically performed in a peer-to-peer fashion, with a central service for registration of components. As a basic communication mechanism, most frameworks rely on remote procedure calls for direct communication and data distribution functionality such as publish and subscribe services. By relying on the communication middleware, the software components of a robot can be distributed among the robot's hardware.

The overhead of communication middlewares results from the pure transmission times of messages over the network and from serialization and message management costs. While this overhead is negligible for infrequent exchanges of individual messages, they constitute a substantial part of the execution times of most robot applications with high modularity and wide distribution of components. Evaluating these costs is possible through measurements within a real system. However, real-world measurements require the system to be deployed, are time-consuming, and only capture one specific hardware assignment of software components.

The communication times and thus the execution times of tasks are strongly influenced by whether frequently communicating components are assigned to the same hardware units, or, if such an assignment is infeasible, by the interconnect between the components. A key goal in the development of robot applications is to minimize communication times to allow for higher loop frequencies within control loops and to assure for subtasks to be executed within certain time frames, e.g., to meet safety requirements.

Developers of robotic applications have mostly relied on intuition and experience to find component assignments to minimize communication costs. With high data rates of sensor values and increasing numbers of software components, analytical tools can be used to find more efficient assignments. By accomplishing a higher frequency in control loops as a result, the preciseness of a robot's grasping task could be increased for example. Therefore, the present paper describes a network simulation for evaluation of execution times and runtime behavior of robotics tasks with different network topologies and component distributions. Our main contributions are as follows:

- Design of a bidirectional coupling between a robotics framework and a network simulation.
- Validation of the performance predictions and optimization of the component topology in an example robotics scenario.

2 FUNDAMENTALS

In this chapter, we present related work and introduce the robotics framework ArmarX.

2.1 Related Work

A number of works describe evaluations of robotics frameworks using microbenchmarks or real-world measurements of example task executions. Shakhimardanov et al. examine the Robot Operating System (ROS) with respect to its scalability (Shakhimardanov et al. 2011). In their benchmark, they use the publish and subscribe service of ROS to simply distribute data from a single publisher to a varying number of subscribers with all components being assigned to a single computer. Therefore, they only examined the local communication overhead without regard for transmission times between hosts. They conclude that ROS scales linearly with the message size, while the message frequency has hardly any impact on the latency, as long as the bandwidth is not worked to capacity. They notice, however, that the number of subscribers unexpectedly influences the latency, which limits the scalability of ROS.

A similar investigation is described by Gijs van der Hoorn (Van der Hoorn 2012). This work compares ROS with Orocos regarding their component interaction latency, jitter on interaction latency and throughput. When comparing a monolithic system to a system with 27 components, both performing the same example task, component-based systems are up to 16 times slower. An analysis reveals that most time is spent in middleware functionality.

A substantial body of research has considered methods for the modeling and simulation of distributed systems. Becker et al. present a meta model to describe the "performance-relevant information of a component-based architecture" (Becker et al. 2007). A simulator is presented that takes an instance of their meta model as input, and outputs workloads and response times generated by simulating the modeled distributed system.

Perumalla et al. propose a simulation framework for large-scale applications based on the Message Passing Interface (MPI) (Perumalla 2010). They designed their simulation to be able to execute existing MPI applications on simulated high-performance clusters. Since the system used for the simulation generally has more limited hardware resources than the simulated system, they modified the MPI runtime library to account for the timing differences between real clusters and the simulated platform. The simulation traps and schedules MPI calls to maintain the order expected in the real system. With the help of test runs they conclude that while there is still room for improvement regarding the performance of their simulation, the runtime predictions are accurate. Similar to their work, our simulation coupling approach traps remote procedure calls.

Some authors focus on accurate models of parallelized applications in high-performance computing environments (Hammond et al. 2009, Rodrigues et al. 2011). While these works provide promising avenues to improve our performance estimations in the future, our present work focuses on the communication costs given by the chosen hardware assignments.

A number of efforts have been undertaken to unify the coupling of simulations and other systems. The goal is to promote reusability and tools for interdisciplinary collaboration. The most prominent framework to compose distributed simulation systems is the High Level Architecture (HLA) (Dahmann et al. 1997). With modularization of simulation domains and the use of interfaces, HLA enables interchangeability of simulation members, so-called federates. Apart from simulations, a federate can also be a hardware system or an arbitrary software application. A similar, more recent approach widely used in industry is the "Functional Mockup Interface" by Blochwitz et al. (Blochwitz et al. 2011). Awais et al. have proposed an execution scheme for hybrid distributed simulations based on models using discrete-event and time-stepped time advancement as well as models based on continuous time (Awais et al. 2013). For the coupling, they rely on HLA and FMI. Similar to their work, our simulation combines a time-stepped robotics framework with a discrete-event network simulation.

Tools such as Shadow (Jansen and Hooper 2011) and the Direct Code Execution facilities in NS-3 (Tazaki et al. 2013) enable the execution of existing applications within simulated network environments. Wegener et al. present an interface to access the traffic simulator SUMO via TCP, which allows for coupling with network simulations (Wegener et al. 2008).

A more ad-hoc approach is given by direct coupling of simulations using custom connections between simulation domains. In real-world examples of direct coupling, socket connections are often used. Mayer et al. provide an overview of coupling mechanisms targeting the network simulator OMNeT++ (Mayer and Gamer 2008). They state that if only application-layer information is required to be sent to the network simulation, direct coupling through a TCP socket is sufficient and requires the least changes to the existing application.

In our work, the main challenges lie in the synchronization between the robotics application and the network simulator, which must be solved independently of whether we rely on a framework such as HLA. Thus, since the intention of the present work is to show the general feasibility and benefit of our approach, we rely on a direct coupling over a TCP socket and leave improvements in reusability and maintainability to future work.

2.2 The Robotics Framework ArmarX

ArmarX is developed by the High Performance Humanoid Technologies Lab at Karlsruhe Institute of Technology and provides several pre-built robot components. One of the core packages is comprised of sensor-actor units that abstract the access to a robot's sensor values and the control of motor units. Further, there are several utility components, e.g., kinematics solvers and motion planners. Developers can create custom software components and orchestrate the components using a statechart mechanism. A dynamics simulator enables modeling a robot's interaction with its physical environment. For this purpose, specific units for sensing and acting inside the simulated environment are provided. The simulator defines its own simulation time, i.e., it is largely decoupled from wall-clock time. The most important use case of ArmarX is to program the humanoid robot series ARMAR. For instance, ARMAR-III (Asfour et al. 2006) is designed to help in known kitchen environments. By combinations of grasping, transportation and placement of objects, it performs tasks like loading of a dishwasher.

For communication between software components, ArmarX's communication middleware relies on the remote procedure call (RPC) framework Ice (ZeroC, Inc. 2018). Ice allows for so-called Ice objects to communicate transparently over a network using RPCs. Calls on remote objects are implemented as function calls on local representations of the remote objects called proxies. Proxies can be obtained from a central registration service. The Ice runtime translates the local calls to remote calls transmitted via the network. For every RPC, an application-layer packet representing the request is created. This packet contains the called object, function and serialized call parameters. ArmarX uses TCP as the transport protocol. On the server, the actual function is executed. If there is a reply, another application layer packet is created and sent to the caller. In ArmarX, every component is implemented as an Ice object providing its functions in an Ice interface. This allows for components to be distributed among different hardware units within a robot. In the following, we refer to the networked hardware units in a robot as *hosts*. Along the RPC functionality, there is also a publish and subscribe mechanism based on topics. Via RPCs, objects can send data to a central server that distributes this data to all objects subscribed to the specified topic. In ArmarX, this service is frequently used to distribute sensor data. The RPC mechanism, as well as the publish and subscribe services represent the communication mechanisms we have to model realistically as a basis for simulating the intra-robot communications.

3 METHODOLOGY

To obtain a meaningful evaluation of intra-robot communication and its effects, a bi-directional coupling is required:

1. ArmarX to network simulator: First, the components' communication behavior needs to be available to the network simulator to evaluate the communication costs. Due to the complex logic in individual components and the large number of components in a typical scenario, modeling each component in a network simulation is not feasible. Hence, we decided to rely on the existing components with only slight modifications to intercept RPCs and send their information to the network simulator through a socket connection.
2. Network simulator to ArmarX: In addition to the pure communication costs, we also want to study the impact of the communication costs of different hardware assignments on the robot's behavior. For instance, longer communication times may lead to delays in control loops and thus affect the precision of robot actions. We achieve a bi-directional coupling by holding back an intercepted RPC within ArmarX, until the corresponding message has passed through the simulated network. Subsequently, within ArmarX, the RPC is transferred and executed physically over the Ice framework.

For our implementation, we used the network simulator OMNeT++ (Varga 2010) and modeled all used ArmarX components as virtual components within our simulation model. Since ArmarX still executes the component logic, the virtual components act as simple senders and receivers of virtual TCP messages representing RPC packets. Further, they account for RPC execution times on the server.

3.1 Synchronization between Simulations

Our goal is to create a coupled simulation in which the messages exchanged between robot components are delayed according to the topology and conditions of the simulated network. Thus, the virtual time between ArmarX and the network simulation must be synchronized.

The virtual time in ArmarX proceeds in a time-stepped fashion, whereas OMNeT++ is based on the discrete-event paradigm, i.e., virtual time advances to the earliest of a set of events scheduled in the simulated future. Thus, the proposed system is a hybrid simulation (Awais et al. 2013). In our implementation, the network simulation manages the progress of the coupled simulation system. The simulation proceeds as follows: the network simulation executes all events up to the current virtual time of ArmarX. Now, the network simulation sends a message to ArmarX to request calculation of the next time step. Subsequently, the network simulation resumes event execution up to the new ArmarX time. During this process, messages are exchanged between the network simulation and ArmarX to trigger the simulated transmission of an Ice message, or to signal the completion of a simulated message transmission.

3.2 Simulation Events

We distinguish three different types of messages transmitted from ArmarX to our network simulation, indicating the following occurrences:

- Routing of an RPC request
- Calculation of an RPC on the server side
- Routing of an RPC response

In Fig. 1, a typical simulation of an RPC within an ArmarX time step is shown. After simulating the transmission of an RPC request, the actual RPC is performed within ArmarX. Subsequently, the calculation of the RPC and the routing of its response can be simulated.

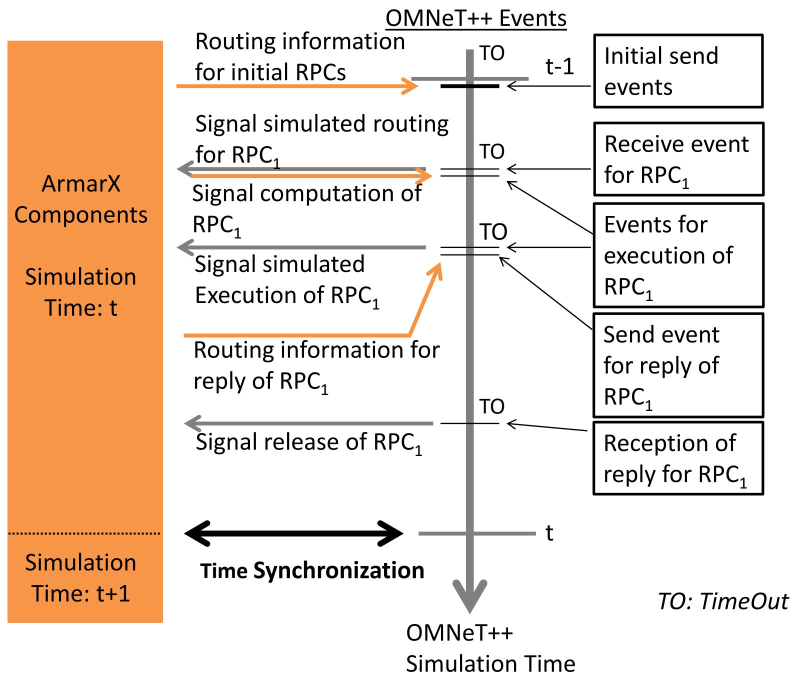
OMNeT++ Events
Routing information for initial RPCs
TO
t-1
Initial send events
ArmarX Components
Simulation Time: t
Signal simulated routing for RPC$_1$
TO
Receive event for RPC$_1$
Signal computation of RPC$_1$
TO
Events for execution of RPC$_1$
Signal simulated Execution of RPC$_1$
Routing information for reply of RPC$_1$
Send event for reply of RPC$_1$
TO
Signal release of RPC$_1$
Reception of reply for RPC$_1$
t
Time Synchronization
Simulation Time: t+1
TO: TimeOut
OMNeT++ Simulation Time

Figure 1: Timeline of the event scheduling for an example simulation step.

Scheduling events corresponding to ArmarX message transfers at the correct virtual time in the network simulation is complicated by the processing times within ArmarX. After calculation of a new ArmarX simulation step, components execute periodic tasks, perform arbitrary calculations and transfer messages. To translate the time required for these steps from wall-clock time to virtual time, a message collection phase is required to allow any message transfers to be signaled to the network simulation before the simulation time proceeds. Further, message collection phases are required at the following stages:

- After the simulated routing of a request, new RPCs may be created as an effect of the current RPC.
- After the simulated execution of an RPC, a response may be created inside ArmarX.
- Finally, after the simulation of a potential response to an RPC, a subsequent RPC may be executed.

During a message collection phase, the network simulation pauses until a configurable timeout in wall-clock time has been reached. As we will show in Section 4.3, the configuration of the timeout enables a tradeoff between the performance and the accuracy of the simulation.

3.3 Measurements

We model network communication costs using the OMNeT++ INET (INET 2018) framework by creating configurable networks of hosts representing the robots' hardware units. ArmarX software components are modeled as TCP applications. The INET framework allows us to realistically estimate the communication costs considering configurable interconnects. To account for processing times within the robot's hosts, measurements of the following quantities are required:

- Execution times of Ice calls.
- Overhead of Ice serialization and de-serialization.
- Message sizes of Ice request and replies.
- Operating system overhead: by default, the INET framework adds 100 nanoseconds of processing time when receiving a TCP packet, representing the overhead of the operating system's network stack. However, real-world overheads substantially exceed this value (Larsen et al. 2009, Emmerich et al. 2014). Relying on values from real-world measurements is particularly important when small messages are exchanged and network stack overheads thus dominate costs of message transfers.

For our simulation, we measured the above quantities during execution of the scenario described in Section 4. During the simulation runs, we determine individual processing times by sampling from the empirical distributions.

4 RESULTS

For evaluation and as a case study, we consider a basic scenario in which ARMAR-III moves its left arm to follow a rectangular trajectory. Despite the simple nature of this task, there are more than 50 components involved. We give a brief overview of the functionality of the scenario and the involved components: the application is specified as a statechart with the main state comprised of a control loop that executes periodically every few milliseconds. In the control loop, the current state of the robot is obtained via an RPC to a *RemoteRobot* component. With the retrieved information about the position of the robot's joints, the target velocity for the robot's hand to reach the end of the current rectangle line can be calculated. This calculation is repeated periodically to account for the effects of external forces such as gravity on the arm. The calculated velocity is then sent via an RPC to the *TCPControlUnit* component, which calculates the target velocities for individual joints. The *TCPControlUnit* runs a separate control loop, also requesting the current state of the robot with an additional call to a *RobotStateComponent* to calculate the target velocities. The velocities are sent to the *KinematicUnit*, which is the actor unit for the robot's joints. Fig. 2 shows a sequence diagram for the RPCs made by the *TCPControlUnit*.

4.1 Validation

To validate our simulation model, we compare simulation runs with distributed executions on physical hardware. The distributed execution was performed on a network of three commodity computers (2x Intel i7-7700, 32 GB RAM; 1x Intel i7-960, 16 GB RAM) connected over a gigabit switch. The required measurements as described in Section 3.3 were performed on the same platform. We ran the simulation

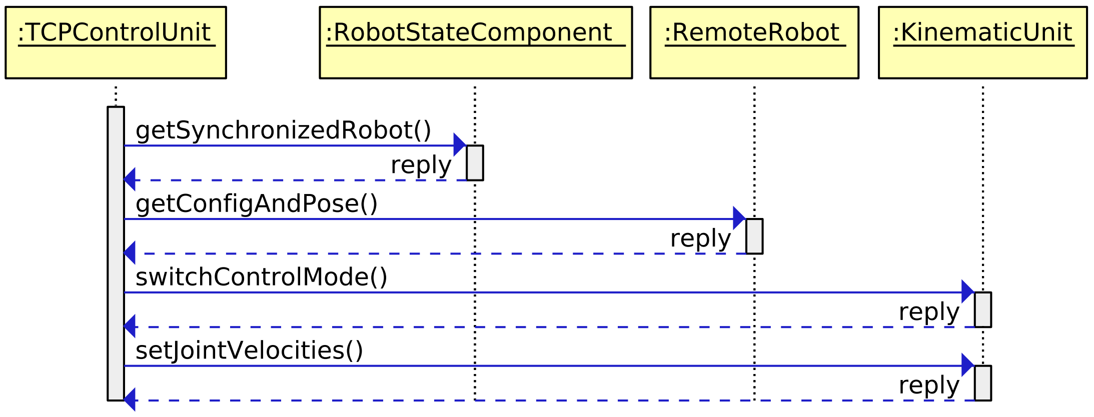

Figure 2: Sequence diagram of RPCs made by the control loop of the TCPControlUnit.

and distributed execution ten times each and measured the time required to complete a cycle for each of the control loops of the *TCPControlUnit* displayed in Fig. 2. In Fig. 3, we show the empirical distributions of the measured cycle times aggregated over all runs for the distributed execution in comparison to the cycle times obtained in the simulation.

It can be seen that the general shape of the empirical distributions matches. In both cases, two peaks about one millisecond apart from each other can be observed. The peaks are caused by the two phases in the scenario's execution: after initialization, until about half of the total scenario time has passed, the robot's arm is not moved. When the *TCPControlUnit* is inactive, it only performs two Ice calls, resulting in lower execution times than with the four calls performed when it controls the robot arm's movement.

Considering the cycle times shown in Table 1, the simulation underestimates the average cycle time for the control loop of the *TCPControlUnit* by about 0.26 ms (17%). We found that the main cause of this difference is the execution cost for process-local computations, which accounts for about 0.2 ms and is not regarded in our simulation. Additionally, we model the Ice framework as a black box, only considering serialization and execution costs of calls. Other tasks contributing to the runtime, such as the scheduling of incoming RPC calls on the server side, might cause additional overhead. This also leads to the slightly lower deviation within and between runs. This difference cannot be seen in confidence intervals due to a lower sample size in simulation runs caused by the robot starting its arm movement faster in simulation.

To validate the correctness of the robot behavior when coupled with the network simulation, we compared the distance of the robot's arm to an ideal rectangular trajectory between distributed executions and simulations. The accuracy depends both on the cycle times and on the delays when propagating sensor

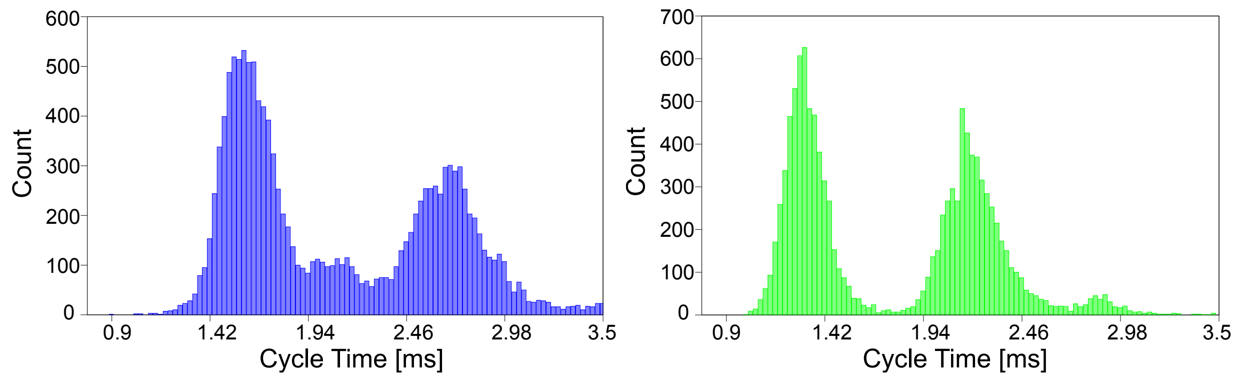

Figure 3: Histograms with cycle times of the *TCPControlUnit* (left: distributed execution, right: simulation runs).

values among the robot components. Comparing the squared errors between simulation and distributed execution shows an average difference of about five percent. Figure 4 contrasts the distance of the hand to the ideal path in a simulation run with a distributed execution.

Overall, we consider the accuracy of our simulation results to be sufficient to examine the timing behavior of the robot. Improvements in accuracy could be achieved by a more detailed modeling of the computations performed by the robot components. In particular, we currently do not consider the effects of the computational load in a component on the processing times.

Table 1: Cycle times of the *TCPControlUnit* for distributed executions and simulations in milliseconds.

	Distributed Execution			Simulation		
Run	Mean	SD	0.95 CI	Mean	SD	0.95 CI
1	2.1146	0.5343	0.0286	1.7862	0.5254	0.0315
2	2.1027	0.6588	0.0344	1.7903	0.5099	0.0312
...						
10	2.0780	0.5728	0.0303	1.8024	0.5159	0.0324
Mean	2.1220	0.6047	0.0317	1.7927	0.5213	0.0321
SD	0.0406	0.0355	0.0017	0.0082	0.0162	0.0008

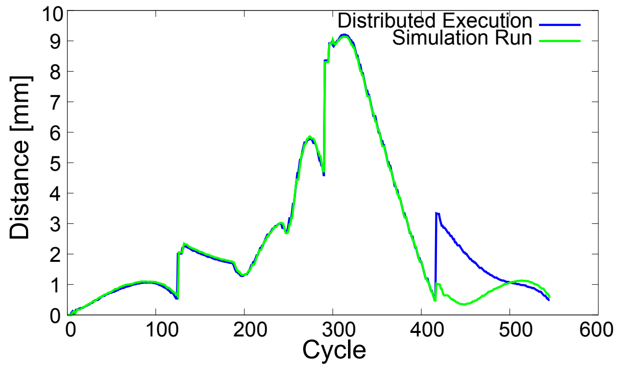

Figure 4: Distance of the robot's arm to the ideal trajectory for a simulation run and a distributed execution.

4.2 Optimization of Hardware Assignment

The main purpose of coupling the robotics framework to a network simulation is to enable optimization of the hardware assignment of components with respect to latencies and bandwidth limitations, e.g., to reduce the end-to-end reaction time to outside events. In the following, we investigate how the coupled simulation can assist robot developers in reducing the time required to complete cycles of the robot's control loops, and the traffic across the inter-component connections.

4.2.1 Cycle Times

First, we focus on minimizing the cycle times of the *TCPControlUnit* by varying the hardware assignment of the involved components displayed in Fig. 2. Naturally, the highest cycle times are achieved with all software components being assigned to different hosts, resulting in the same results as listed in Table 1. As expected, the lowest cycle times are achieved by assigning all components to the same host. This approach reduces cycle times by more than half to about 0.88 ms. The second fastest cycle times are achieved by isolating the *KinematicUnit*, with average cycle times of 1.13 ms, followed by isolating the component providing robot information, with cycle times of 1.56 ms.

4.2.2 Network Traffic

We conducted a second experiment using the same scenario as above, now focusing on minimizing the bandwidth usage when increasing the number of hosts the software components are assigned to. Since the lowest bandwidth usage will always be achieved when assigning all components to the same host, we postulate that the software components are distributed equally among the hosts. Such constraints could be given in a system with limited bandwidth and processing power. To minimize the bandwidth consumption, we computed the minimum edge cut in a communication graph created from simulation data. The edge cut was computed heuristically using Metis (Karypis and Kumar 1995). The results can be seen in Fig. 5. The proposed hardware assignment with two hosts requires only 2.6 MBit/s of bandwidth. Up to six hosts, the total throughput rises linearly to about 21 MBit/s. With seven hosts and beyond, the bandwidth usage grows more slowly, since fewer and fewer pairs of interacting components are separated. With twelve hosts, the bandwidth usage is at 32 MBit/s, which is about a third of the extreme case of assigning each software component to a separate host.

We now consider the cycle times achieved when minimizing the bandwidth usage. When increasing the number of hosts, the cycle times increase sharply due to the separation of components with frequent interaction. The variation in cycle times between eight and twelve hosts is not reflected in the bandwidth usage, demonstrating that minimizing one of these quantities does not necessarily minimize the other.

We can show that the hardware assignment suggested by the minimum edge cut in the communication graph does not minimize the cycle times. To this end, we chose the configuration with five hosts and manually swapped the hardware assignment of two components, which was sufficient to group all components that crucially affect the cycle times to the same host. A simulation run with this hardware assignment results in average cycle times of 0.89 ms, which is 28% lower than the averaged 1.23 ms in Fig. 5.

The considered scenario was small enough to perform the above experiments by manual adaptation of the hardware assignments. To cover more complex architectures, the experimentation can be automated to systematically steer the configuration towards an optimal hardware assignment in a simulation-based optimization process (Gosavi 2003).

4.3 Performance

The performance of the coupled simulation is most strongly affected by the configurable duration of the timeouts for collecting messages from ArmarX. Fig. 6 shows the simulation runtime with different timeouts, contrasted with the error in the prediction of cycle times. The error was calculated by comparing average cycle times with those of a simulation run utilizing timeouts of 20 milliseconds. Larger timeout durations lead to higher execution times but more accurate results, since short timeout durations may cause messages from ArmarX to be missed during message collection and thus postponed in simulation time.

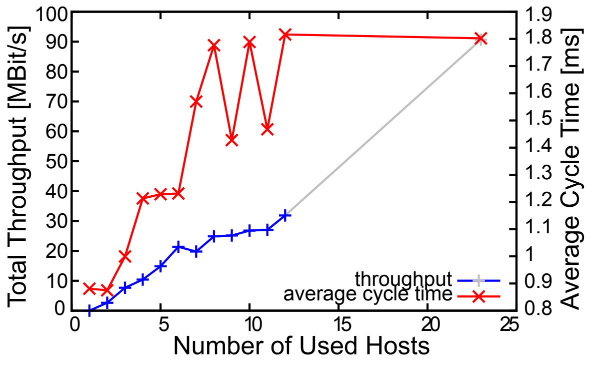

Figure 5: Bandwidth usage and cycle times with increasing number of hosts.

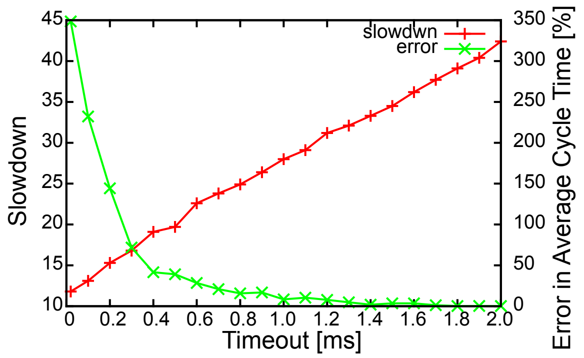

Figure 6: Performance of the simulation dependent on used timeout window.

A smaller amount of runtime is spent on computing the next time step within ArmarX, and on event handling within OMNeT++. We recorded an average of 3.69 ms for requesting a new ArmarX time step and receiving its confirmation, which allows the simulation time to proceed by 1 ms. Per second of simulation time, about 550.000 events were handled by OMNeT++.

5 CONCLUSION AND DISCUSSION

We presented a coupling of a robotics framework with a network simulator that enables realistic evaluations of intra-robot communication networks, taking into account the execution times of remote procedure calls. Predicting the runtime of robot tasks can help researchers and developers to optimize applications and the hardware assignment of software components. Although our simulation currently targets the ArmarX framework, the general approach is applicable to other robotics frameworks that allow for interception of message transmissions at the sender.

Our prototype implementation still has a number of limitations that we intend to tackle in the future to allow for a complete integration within the robotics framework ArmarX:

- Performance: Since there are no guarantees about future messages sent by ArmarX, timeouts are required to achieve a low probability of missing a message. As a consequence, the slowdown of our simulation compared to local executions is still quite high. Eliminating timeouts entirely would require substantial modifications to the ArmarX codebase. Some performance improvements may be achieved by separately tuning the timeouts for the three phases of a remote procedure call.
- Simulation of local computations: The proposed network simulation translates the execution times of remote procedure calls to simulation time. Computations that are independent of remote procedure calls are executed according to wall-clock time and not considered in the simulation. Thus, the simulated overall execution times are typically lower than in real-world executions. Modeling the timing behavior of local computations will require more intrusive adaptations to ArmarX.
- Evaluation of further scenarios: Our evaluation and validation was performed with respect to a comparatively simple movement of a robot arm, which already involved more than 50 components. More complex tasks could be considered to evaluate and optimize larger intra-robot architectures.

Lastly, we intend to further improve our simulation model. Most importantly, computational costs could be modeled by factoring in configurable CPU specifications and the simulated system load. Additionally, by modeling the costs for processing of messages throughout the network stack at more detail, the accuracy of the predictions could be improved.

ACKNOWLEDGEMENT

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme. This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre Invasive Computing (SFB/TR 89).

REFERENCES

Asfour, T., K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann. 2006. "ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control". In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 4th–6th December, Genova, Italy, 169–175.

Awais, M. U., P. Palensky, W. Mueller, E. Widl, and A. Elsheikh. 2013. "Distributed hybrid simulation using the HLA and the Functional Mock-up Interface". *Industrial Electronics Society, IECON*:7564–7569.

Becker, S., H. Koziolek, and R. Reussner. 2007. "Model-Based Performance Prediction with the Palladio Component Model". In *Proceedings of the 6th International Workshop on Software and Performance*, 5th–8th February, Buonos Aires, Argentina, 54–65.

Blochwitz, T., M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, S. Wolf, and C. Clauß. 2011. "The Functional Mockup Interface for Tool Independent Exchange of Simulation Models". In *Proceedings of the 8th International Modelica Conference*, 20th–22nd March, Dresden, Germany, 105–114.

Bruyninckx, H. 2001. "Open Robot Control Software: the OROCOS Project". In *IEEE International Conference on Robotics and Automation (ICRA)*, 21st–26th May, Seoul, South Korea, 2523–2528.

Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly. 1997. "The Department of Defense High Level Architecture". In *Proceedings of the Winter Simulation Conference*, edited by S. A. et al., 142–149. Piscataway, New Jersey: IEEE.

Elkady, A., and T. Sobh. 2012. "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography". *Journal of Robotics* 2012.

Emmerich, P., D. Raumer, F. Wohlfart, and G. Carle. 2014. "A Study of Network Stack Latency for Game Servers". In *Proceedings of the 13th Annual Workshop on Network and Systems Support for Games*, 4st–5th December, Nagoya, Japan, 15.

Gosavi, A. 2003. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Norwell, MA, USA: Kluwer Academic Publishers.

Guizzo, E., and E. Ackerman. 2012. "How Rethink Robotics Built its New Baxter Robot Worker". *IEEE spectrum*:18.

Hammond, S. D., G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. 2009. "WARPP: A Toolkit for Simulating High-Performance Parallel Scientific Codes". In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2nd–6th Rome, Italy, 19:1–19:10.

INET 2018. "INET Framework". https://inet.omnetpp.org/, accessed March 6th, 2018.

Jansen, R., and N. Hooper. 2011. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". Technical report, Minnesota University, Department of Computer Science and Engineering.

Karypis, G., and V. Kumar. 1995. "METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0". Technical report, University of Minnesota, Department of Computer Science.

Larsen, S., P. Sarangam, R. Huggahalli, and S. Kulkarni. 2009. "Architectural Breakdown of End-to-End Latency in a TCP/IP Network". *International Journal of Parallel Programming* 37(6):556–571.

Mayer, C. P., and T. Gamer. 2008. "Integrating Real World Applications into OMNeT++". Technical report, University of Karlsruhe, Institute of Telematics.

Metta, G., P. Fitzpatrick, and L. Natale. 2006. "YARP: Yet Another Robot Platform". *International Journal of Advanced Robotic Systems* 3(1):8.

Perumalla, K. S. 2010. "$\mu\pi$: A Scalable and Transparent System for Simulating MPI Programs". Technical report, Oak Ridge National Laboratory (ORNL), Center for Computational Sciences.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. 2009. "ROS: an Open-Source Robot Operating System". In *ICRA Workshop on Open Source Software*, 17th May, Kobe, Japan, 5.

Rodrigues, A. F., K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. 2011, March. "The Structural Simulation Toolkit". *SIGMETRICS Performance Evaluation Review* 38(4):37–42.

Shakhimardanov, A., N. Hochgeschwender, M. Reckhaus, and G. K. Kraetzschmar. 2011. "Analysis of Software Connectors in Robotics". In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 25th–30th September, San Francisco, California, 1030–1035.

Tazaki, H., F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous. 2013. "Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments". In *Proceedings* *of the 9th ACM conference on Emerging Networking Experiments and Technologies*, 9th–12th December, Santa Barbara, California, 217–228.

Vahrenkamp, N., M. Wächter, M. Kröhnert, K. Welke, and T. Asfour. 2015. "The Robot Software Framework ArmarX". *it-Information Technology* 57(2):99–111.

Van der Hoorn, G. 2012. "Performance Analysis of Middleware for Component Based Robot Control Software". Master's thesis, Delft University of Technology, Department of Software Technology.

Varga, A. 2010. *OMNeT++*, 35–59. Berlin, Heidelberg: Springer.

Wegener, A., M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux. 2008. "TraCI: an Interface for Coupling Road Traffic and Network Simulators". In *Proceedings of the 11th Communications and Networking Simulation Symposium*, 14st–17th April, Ottawa, Canada, 155–163.

ZeroC, Inc. 2018. "Ice – Comprehensive RPC framework". https://zeroc.com/products/ice, accessed March 6th, 2018.

AUTHOR BIOGRAPHIES

DANIEL KRAUSS received his M.Sc. in Computer Science in 2017 from KIT (Karlsruhe Institute of Technology). His email address is danielrkrauss@online.de.

PHILIPP ANDELFINGER is a research fellow at TUMCREATE Ltd and Nanyang Technological University (NTU), Singapore in the group of Prof. Wentong Cai. He received his diploma and Ph.D. in Computer Science in 2011 and 2016 from Karlsruhe Institute of Technology (KIT), Germany. His research focuses on parallel and distributed simulation in heterogeneous hardware environments, agent-based simulation, and network simulation. His email address is pandelfinger@ntu.edu.sg.

FABIAN PAUS received his M.Sc. in Distributed Information Systems from the WHS (Westfälische Hoschule, Bocholt) in 2016. He is employed as a research scientist and Ph.D. student at the Institute for Anthropomatics, Karlsruhe Institute of Technology (KIT). His major research interests include active perception as well as manipulation in the context of humanoid robotics. His email address is paus@kit.edu.

NIKOLAUS VAHRENKAMP received his Diploma and Ph.D. degrees from the Karlsruhe Institute of Technology (KIT), in 2005 and 2011, respectively. He was a postdoctoral researcher at the Institute for Anthropomatics, Karlsruhe Institute of Technology (KIT) and worked on software development, grasping and mobile manipulation for the humanoid robots of the ARMAR family. He is the author of over 40 technical publications, proceedings and book chapters. His research interests include humanoid robots, motion planning, grasping and sensor-based motion execution. His email address is vahrenkamp@kit.edu.

TAMIM ASFOUR is a full Professor at the Institute for Anthropomatics, Karlsruhe Institute of Technology (KIT). He is chair of Humanoid Robotics Systems and head of the High Performance Humanoid Technologies Lab (H2T). His current research interest is high performance humanoid robotics. He is developer and leader of the development team of the ARMAR humanoid robot family. He has been active in the field of Humanoid Robotics for the last 14 years resulting in about 150 peer-reviewed publications with focus on engineering complete humanoid robot systems including humanoid mechatronics and mechano-informatics, grasping and dexterous manipulation, action learning from human observation, goal-directed imitation learning, active vision and active touch, whole-body motion planning, system integration, robot software and hardware control architecture. He received his diploma degree in Electrical Engineering in 1994 and his PhD in Computer Science in 2003 from the University of Karlsruhe. His email address is asfour@kit.edu.