Adaptive Model-Based Monitoring for Robots

Dominik Kirchner and Kurt Geihs

Abstract Continuous and comprehensive monitoring is a key requirement for reliable failure detection. However, the overhead of the observation process conflicts with the limited resources of a robot platform. Therefore, robot monitoring faces high efficiency requirements. This defines a trade-off between comprehensive observation and monitoring resource overhead. In this paper, we propose an adaptive, model-based monitoring approach that addresses this trade-off. We specify an individual monitoring configuration in an abstract system model to focus the observation on expressive state aspects. Moreover, we introduce adaptivity to further improve the efficiency of the monitoring process. To evaluate this efficiency, we compare our approach with a reference monitoring system. Due to our results, we are confident that the proposed approach significantly reduces the resource overhead.

1 Introduction

Advances in autonomous robotic research are increasingly applied in multiple areas spanning from industrial production to medical assistance systems [1]. This results in increasingly complex and error-prone system designs [2]. However, the ability to autonomously manipulate their physical environment demands a high reliability requirement of these robots. Therefore, reliability and safety pose one of the major challenges in the field.

To counteract performance-critical events in a running robot system, automatic failure recovery is required at runtime. Therefore, robots require continuous and comprehensive system state monitoring to support reliable failure diagnosis and recovery. The design objective of such a monitoring system is to provide comprehensive and expressive state information with minimal delay. Unfortunately, monitoring

D. Kirchner (✉) · K. Geihs
Distributed Systems Group, University of Kassel, 34121 Kassel, Germany
e-mail: kirchner@vs.uni-kassel.de

K. Geihs
e-mail: geihs@vs.uni-kassel.de

E. Menegatti et al. (eds.), *Intelligent Autonomous Systems 13*,
Advances in Intelligent Systems and Computing 302,
DOI 10.1007/978-3-319-08338-4_4

capabilities are often an afterthought in robot development [3]. Such capabilities are rarely initially included in the system architecture [4]. However, late integration often causes high integration efforts. Hence, monitoring support is omitted, if the integration efforts are considered too high. This complicates the monitoring design objective to provide comprehensive state information. In order to contribute to a comprehensive monitoring, the integration efforts should be reduced. Moreover, computational resources for autonomous robots are constrained severely. Acting in real-world environments already consumes a significant amount of resources, e.g. for sensor data analysis or action execution. As a consequence, system components not directly required for robot operation, like the monitoring, face high efficiency requirements to avoid interferences with robot operation [4]. This efficiency requirement conflicts with the requirement of expressive and frequent system observation [5].

In summary, the design of a monitoring system requires comprehensive, expressive, and frequent state observation, while simultaneously providing acquisition efficiency and ease of integration. This forms a trade-off between monitoring quality and resource overhead. In order to contribute to this challenge, we propose an adaptive, model-based monitoring for robot systems. The key contributions of our proposed approach are:

Model-based Configuration We use an abstract model to specify a priori knowledge of the robot design and the application domain to specialize the monitoring.The proposed abstract model enables the monitoring to focus on relevant state aspects tailored to the application context.This eases the monitoring integration and contributes to an individual configuration of comprehensive and expressive state observation.

Adaptation We use adaptation to adjust the monitoring behavior to the current system context. This allows to automatically fine-tune the acquisition process to react to performance-critical state changes. This contributes to the efficiency requirement.

The paper is structured as follows: In Sect. 2, we discuss related work for robot monitoring. We introduce an abstract robot model to specify domain knowledge in Sect. 3. Section 4 presents our monitoring solution, like an individual monitoring configuration in Sect. 4.1 and adaptation support in Sects. 4.2 and 4.3. The evaluation in Sect. 5 presents simulation results in a hypothetical robot scenario. We conclude the paper in Sect. 6 and give an outlook on future work.

2 Related Work

Only little attention has been spent on the challenge of reliability in the robot domain [2], and even less contributions have been published on the essential part of robot monitoring. In practice, basic monitoring support is often found as part of a robot development framework. However, this support is often a side issue, providing that it exists at all [6].

A commonly used development framework, which supports monitoring, is the Robot Operating System (ROS) [7]. ROS provides monitoring features by the *diagnostic updater* and *node monitoring* components. Both provide development support in terms of libraries and language-specific utilities. These features have to be manually integrated in the source code and specially adjusted to the application. As a consequence, the realization of a system monitoring is a highly manual, time consuming, and application-specific task, which requires profound expert knowledge. The implementation of the monitoring scope and the resource efficiency of the acquisition is completely left to the developer.

Another robot development framework is CLARAty [8]. CLARAty provides extensive support for task execution monitoring within a robot system. Based on the Task Description Language (TDL) [9], CLARAty models the monitoring behavior, the acquisition, and processing of task-related data in accordance to the task hierarchy. The total state of an object is tracked and can be queried on demand. The selection of the state data and the observation mechanism are not further specified and therefore expected to be done case-specifically. However, the main focus of this monitoring is the observation of task execution. Observation of the operational state for components, like heartbeat signals or resource usage checking, is not addressed. Furthermore, the work does not consider efficiency of the state tracking. Low-level monitoring of components is also not further specified and therefore expected to be handled manually.

Similar to CLARAty, CWave [5] provides a development framework build on formal semantics (Instrumented Logical Sensor System, ILSS). In this project, the goal is to achieve a comfortable way to compare different versions of a sensor system based on online and offline quality measurements. This is accomplished by embedded functionalities for performance measurements, like runtime checks, component state observers, or embedded test routines, in each system component. However, this instrumentation is embedded in the component during the development process. Observation of components not developed within this framework is not supported. Moreover, the issue of efficient system observation is not further covered.

In [4, 10] the authors present a model-based diagnosis for robot systems. Based on a software model, a failure is detected when the monitored value exceeds its specified limits, like timing constrains in the communication [11]. This monitoring is done on a regular but static basis for a constant set of manually specified observations. Therefore, the efficiency requirement of an observation mechanism do not scale well with the system size.

In contrast, our work focuses on efficient system observation. The monitoring configuration is specified in a generic robot model. Similar to TDL, this model captures the system structure and allows the user to define an individual monitoring configuration. Additionally, we address the efficiency requirement by an adaptive observation mechanism. Based on that model, application-specific adjustments can be done without changes on the code level of components. Therefore, we argue that this monitoring can be applied even to an already existing robot system, with little integration effort [12].

3 System Model

In this section, we present a model to specify a priori knowledge of the application domain and the system design. Therefore, we first describe a modeling formalism and later specialize this to the robot domain.

Our modeling formalism is based on [13]. The most general concept in this formalism is the *System*. A system is an entity that interacts with other entities, i.e. other systems including hardware, software, and the physical world. The relation between systems creates a structure $\mathcal{M}$, as seen in Fig. 1. This structure is defined as a tuple $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S} := \{s_0, s_1, \dots, s_N\}$ denotes a transitive set of systems s_i and $\mathcal{R}$ defines a set of relations between the systems. Each relation is a 2-element subset of $\mathcal{S}$ where the elements are the participating systems. The relations are undirected $(s_i, s_j) = (s_j, s_i)$ and defined as:

$$\mathcal{R} := \{(s_i, s_j) \mid i \neq j\}. \tag{1}$$

As a consequence, relations are symmetric, irreflexive, and intransitive. The environment $\mathcal{E}_{s_g}$ is specified as a set of systems that interact with s_g.

$$\mathcal{E}_{s_g} := \{s_i \mid (s_g, s_i) \in \mathcal{R}\}. \tag{2}$$

The boundary $\mathcal{B}_{s_g}$ of a system is defined as the common frontier between the given system s_g and its environment $\mathcal{E}_{s_g}$.

$$\mathcal{B}_{s_g} := \{r_i \mid r_i = (s_i, s_g), s_i \in \mathcal{E}_{s_g}\}. \tag{3}$$

Components of a top-level system $s_\top$ are understood again as systems s_i and form a structure. This extends the formal structure and defines a recursion $s_i \in s_\top$.

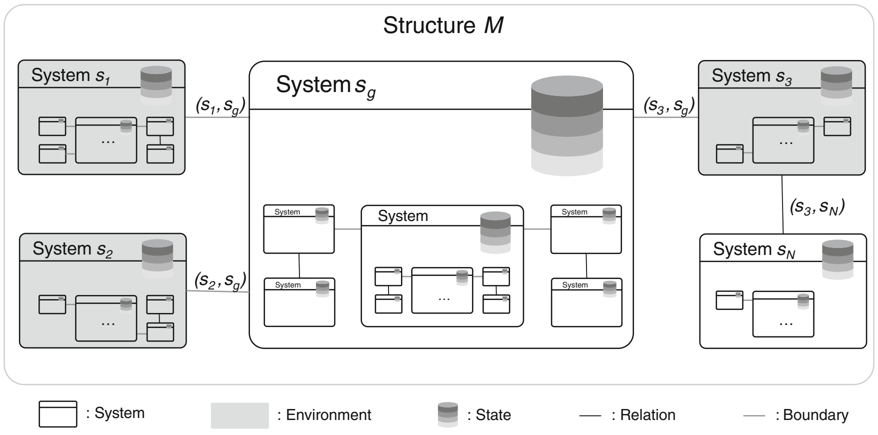

Fig. 1 Recursive system formalism

The recursion stops when a system is considered as atomic $\nexists s_i(s_i \in s_\perp)$. An atomic system $s_\perp$ is considered as a system with no distinguishable structure, or a structure that is of no further interest. In our context, we understand an *atomic* system as an operating system process or a distinct hardware module.

In order to reflect the current context of a system, we model the system state. The $state$ of a technical system subsumes all relevant aspects of the operation in a time period. The state includes the aspects of computation $comp$, stored information $data$, interconnection $flow$, and physical conditions (resources) res. Therefore, we define a state as a quadrupel of these aspects $state := (comp, data, flow, res)$. Elements of this quadrupel are in turn sets of characteristic aspects, e.g. CPU or memory usage for the computation. We denote single elements of this state as a characteristic $charac$

$$\mathcal{C}_{s_i} := \{charac \mid charac \in state\}. \tag{4}$$

The total state of a system $state_{s_i}$ comprises all states of subsystems of s_i:

$$state_{s_i} := \{state_0, state_i, ...state_M\} \tag{5}$$

Based on this general formalism, we derive a model for the robot domain. Therefore, we represent the structure $\mathcal{M}$ as a hierarchical layered tree, as depicted in Fig. 2. Each layer defines an abstraction level of the system. We structure the abstraction levels in the layers: organization, members, skills, and realization. In order to support large scale scenarios, the first layer models organizational details, like group hierarchies. The second layer contains members of a specified organization, like robots.

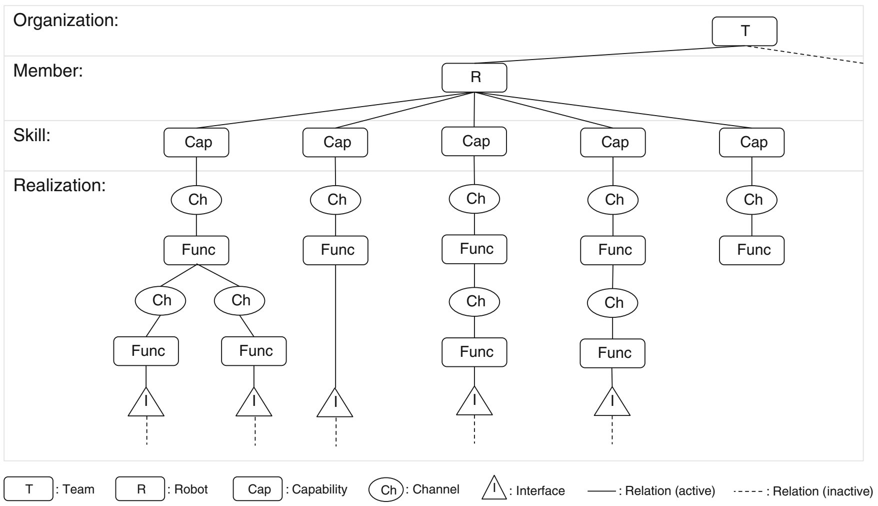

Fig. 2 Layered robot model

Each member is modeled by its skills. In order to realize such skills, we model the required information processing of the involved systems on the realization layer. Relations between these systems are represented as additional elements in the model.

We limit the focus in this paper to a single robot system. Organizational details are not addressed further. Furthermore, we name systems in respect to their layers. We refer the top-level system as *Robot*, while systems that realize skills are called *Functionalities*. Relations between functionalities are denoted as *Channels*. This information processing chain ends with an *Interface* element that marks the end of the software domain layer and the beginning of the hardware layer.

4 Monitoring

The design objective of the monitoring is to provide comprehensive, expressive, and frequent state observations with minimal resource overhead. Based on the formalism of Sect. 3, we define a monitoring task as a method *mon* to provide observations on a subset of characteristics $\mathcal{C}_{s_i}$ of the total state.

$$mon : \mathcal{S} \rightarrow \mathcal{C}_{s_i}. \tag{6}$$

A realization, which concurs with our design objectives, needs to specify an expressive subset of $\mathcal{C}_{s_i}$ and an efficient acquisition process. To specify these expressive characteristics, we propose an individual configuration in Sect. 4.1. To comply with an efficient acquisition process, we suggest two types of adaptations. The first type adapts the observation in respect to the current operational context, the second in respect to the current robot task (see Sects. 4.2 and 4.3). We differentiate two acquisition types for state observation. In the first type, components are extended with monitoring features to provide state information by their own (see Sect. 2, ROS [7]). A heartbeat signal is a common example of this, where a component continuously sends messages to prove its liveliness. We refer to this as *active monitoring*. This, however, often requires code changes and causes manual integration efforts. Additional to this common acquisition type, we propose a gathering concept that focuses on the collection of general, characteristic information of a component, like resource usage, or communication properties. We refer to this as *passive monitoring*. In contrary to active monitoring, passive monitoring needs no support from a component. Instead, it interfaces external sources, like the operating system, to gather relevant data. This enables a realization with a separated, independent module. As a result, integration of passive monitoring is facilitated, especially for already existing systems [12]. Nevertheless, the expressiveness of these external observations seems sufficient for the diagnosis task, as first results from [14] indicate.

In order to exemplify our proposed concepts, we introduce a hypothetical search-and-collect scenario. Here, the goal of a robot is to explore the environment, find items of interest, and transport them back to a certain location. Furthermore, we subdivide this scenario in the tasks: explore the environment (search task), collect the item (collection task), transport the item to the defined location (transport task), and unload the item (dump task). The envisioned robot provides the skills to move, localize itself, detect objects, manipulate objects, and acoustically report its status. In respect to the proposed system model, this robot is modeled as a tree node *R*, while each skill is presented as a capability (*Cap*, see Fig. 2). The realization of the skills are left as an unspecified composition of atomic system components (*Func*) and interactions (*Ch*).

4.1 Individual Configuration

Modular robot design consists of multiple components. Each realization of a component is geared to a special purpose in this system design [11]. Hence, we assume that characteristic observations are also component-specific. In order to find an expressive set of observations, individual configuration of the characteristics is required. We further define the extent of this set as a monitoring scope $\mathcal{SC}_{robot} \subseteq \mathcal{C}_{robot}$. In practice, complete observation of the robot's total state $\mathcal{SC}_{robot} = \mathcal{C}_{robot}$ is too resource-intensive. Therefore, we limit the scope to a reduced subset $|\mathcal{SC}_{robot}| \ll |\mathcal{C}_{robot}|$. We individually annotate elements in the model with a configuration element *Characteristic*, which determines the scope of the element. The cumulated characteristics define the individual scope $\mathcal{SC}_{robot} = \{\mathcal{SC}_{s_i} \mid s_i \in \mathcal{S}\}$.

Beside an individual monitoring scope, the degree of importance also varies for components in the robot model. For example, the acoustic output skill provides little relevance for the scenario completion, so the observation should be minimized. Therefore, we introduce an additional configuration element in the model, named *Priority*, to reflect the individual importance for each element. This local priority is individually specified in respect to its abstraction level. For example, a motor driver component is considered to have a high priority for the realization of the motion skill, irrespective of whether or not the motion skill is currently needed. As a consequence, priority specifications on the realization layer are independent of the designated application in a scenario. Application-specific aspects are again locally modeled on higher layers. We define this local priority as:

$$prio_l : \mathcal{S} \to \mathbb{R}. \tag{7}$$

In order to obtain the resulting, global priority $prio_g$, we introduce a hierarchical priority rating in the model.

$$prio_g(s_{i,n}) = prio_g(s_{i,n-1}) + prio_l(s_{i,n}). \tag{8}$$

The priority rating forms a recursion over the defined layers n. We denote a system s_i on the layer n as $s_{i,n}$, e.g. the top-level system is defined as $s_\top = s_{i,0}$.

4.2 Operational Adaptivity

On top of the individual configuration, we incorporate operational knowledge to further reduce the resource usage. Operational knowledge specifies characteristic state values of system components. These values are taken as references for normal operation. High deviations to the reference are considered as indication for an unusual operational state and result in intensified observation, while low deviations decrease the monitoring.

More specifically, the task of the proposed adaptive monitoring is to adjust the acquisition process in respect to the global priority of a component. We control the acquisition by a set of parameters p_i, further denoted as a monitoring configuration $mc = (p_1, p_2, \ldots, p_N)$. These parameters are domain specific and range from simple observation interval length to more sophisticated analysis settings like model estimation. The adaptation *adapt* changes these parameters in respect to the current operational context. We describe the supported adaptations by a limited set of monitoring configurations $\mathcal{MC}_{i,char} := \{mc_{i,char}\}$ specific for a system s_i and its defined characteristics $char$. Formally, the adaptation maps a priority to a monitoring configuration:

$$adapt : prio_g(s_{i,n}) \rightarrow \mathcal{MC}_{i,char}. \tag{9}$$

This initially defines an individual acquisition process. However, the operational situation may change dynamically. In such a situation the monitoring configuration should be dynamically adapted to the current operational situation. For example, a suspicious component state, like a continuously rising motor temperature, should yield to intensified observation. Therefore, we extend the *Characteristic* element in the model to additionally capture representative information on the operational state. An example for such information are typical values of a characteristic, like an average value, or variations. We model these typical values in a *Property* and add it to the characteristic.

Based on a given property $prop_i$, we define a metric $dev = ||o, v||_{prop_i}$ to determine the deviation between a current observation o and the representative value v. For example, we assume that motor temperatures fluctuate between a minimum and a maximum value. This defines a range of typical observations. Here, the fluctuations are expected to be symmetrical, hence the middle of the interval is considered as the typical value. Small deviations represent a good fit to the expected value and reduce the monitoring process, whereas large deviations indicate a suspicious state and intensify the monitoring process. This dynamic adaptation is controlled by the function $adapt_{dyn}$. We extended this function by the deviation dev to capture the current operational state.

$$adapt_{dyn} : prio_g(s_{i,n}) \times dev \rightarrow \mathcal{MC}_{i,char}. \tag{10}$$

In order to address the time-varying nature of a robot system, this adaptation process is evaluated in runtime to continuously adapt the monitoring to the current situation.

4.3 Task-Related Adaptation

On top of the operational adjustments, we consider task-related adaptation. As exemplified in our search-and-collect scenario, robot missions are often structured in tasks $\mathcal{T} := \{t_0, t_1, \ldots, t_N\}$. Each task t_i may require a different subset of robot skills $t_i \subset \mathcal{SK} := \{s_{0,n}, s_{1,n}, \ldots, s_{N,n}\}$ where n specifies the skill layer. Static observations of the complete robot may result in the observation of currently inactive skills. For example, in the search task of the scenario, the robot does not use its manipulation skill. Hence, monitoring of components related to item manipulation are temporarily not required. This provides potential to further reduce resource usage of the monitoring without losing expressiveness of the observation. Therefore, we adjust the monitoring scope in respect to the current task of the robot $\mathcal{SC}_{robot,t} := \{\mathcal{C}_s \mid s \in t_i\}$.

In order to provide this information, we add task information to the skills in the model. Therefore, we extend the *Priority* element to a list of task-dependent priorities. More specifically, we add a subset $\mathcal{T}_{s_i} \subset \mathcal{T}$ to each skill $s_{i,n}$ to describe in which tasks a skill is required. Hence, we extend the definition of the local priority, given in (7):

$$prio_{l,t} : \mathcal{S} \times \mathcal{T} \rightarrow \mathbb{R}. \tag{11}$$

As a consequence, the global priority rating, given in (8), is also revised:

$$prio_{g,t}(s_{i,n}) = prio_{g,t}(s_{i,n-1}) + prio_{l,t}(s_{i,n}, t_j). \tag{12}$$

This leads to the extended task-related adaptation:

$$\text{adapt}_{\text{dyn, t}} : prio_{g,t}(s_i, n) \times dev \rightarrow \mathcal{MC}_{i,char}. \tag{13}$$

Note that the necessary task information t_j is expected to be provided by the control system of the robot. Without such a notification, the task-related adaption is not directly applicable (Fig. 3).

In summary, we propose an adaptive, model-based monitoring based on an individual and task-related scope $\mathcal{SC}_{robot,t}$ to focus the monitoring to expressive state aspects. In addition, we define an adaptive acquisition progress $\text{adapt}_{\text{dyn, t}}$ in relation to the task and the operational context to reduce the resource overhead.

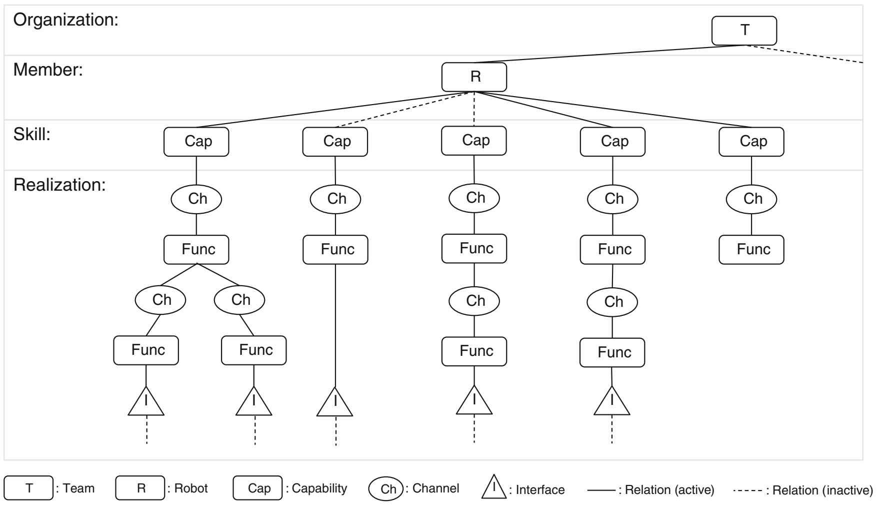

Fig. 3 Task-related configuration for a hypothetical robot in the search-and-collect scenario (transportation task, see Table 1). The skills (beginning from *left*) are: drive, grab, detect, localize, and control. Currently, inactive skills are marked as *dashed lines*

5 Evaluation

To determine the efficiency of our proposed monitoring solution, we conducted a number of simulation experiments. More specifically, we evaluate the quality of the monitoring in terms of the properties: efficiency and scalability. The efficiency property is defined as the ratio of resource usage between a constant and static reference monitoring and our proposed monitoring. The scalability property, in addition, captures the resource usage of the approach in respect to the system size. In order to ground this evaluation, we evaluate our approach in respect to two versions of this reference monitoring. To frame the range of resource usage, we apply a pessimistic and an optimistic version. In these reference monitoring configurations, we do not apply individual configuration nor adaptation. While the optimistic setting assumes a low failure probability and therefore represents a minimalistic acquisition process, the pessimistic setting anticipates a high failure rate and performs an intensive observation. We separately evaluate the effectiveness of our approach in respect to the proposed concepts from Sect. 4. Therefore, we increase the applied concepts in each instance, starting from an individual configuration, over operational adaptation, to task-related adaptation. In order to evaluate the scalability, we additionally vary the size of the system. Therefore, we increase the number of components in the simulated robot system.

5.1 Simulation Setup

The setup of the simulation consists of a simulated robot system, a component to control the size of the system, and the monitoring system. The simulated robot consists of multiple, generic components to represent modular data processing and intercomponent communication of most modern robot systems. These components connect to each other, process data, and simulate resource usage. In our simulation, we concentrate on the observation of the software layer.

We change the system size by generating robot models with a varying number of functionalities on the realization layer. These models are interpreted by the Capability and Reliability Manager (CARE) [12, 15], which also implements the monitoring. In our evaluation, we focus on passive monitoring. Therefore, the monitoring scope is based on general, external state information, like CPU or memory usage. Additionally, CARE monitors the intercomponent communication, like message frequency or message content. In detail, the observed characteristics in this simulation are the CPU usage, memory usage, number of threads, and message frequency.

5.2 Simulation Configuration

In the simulation, we implement our search-and-collect example scenario. Our simulated robot provides the skills: drive, grab, object detection, localization, and control. The number of components to realize the skills, except the control skill, are adjusted to scale the size of the simulated robot. Every component is parameterized to generate a constant resource usage of 3 % CPU usage and 25 MB memory. We subdivide this scenario in the tasks: search, collect, transport, and dump.

In our simulation, the priorities are restricted to the discrete range $[0, 1, \ldots, 10]$. Accordingly to the dynamic environment of a real-world robot application, we implement short observation intervals ranging from 100 ms for priority 10–5000 ms for priority 0 ([5000 ms, ..., 590 ms, 100 ms]). The functionalities are equally prioritized with one, while the skills determine the varying priorities. In Table 1, the first row presents the task-related prioritization of the skills. Empty entries mark skills not required for the task. The following rows present the priorities for the pessimistic, the individual configuration, the operational adaptation, and the optimistic monitoring. The scope of the monitoring is identically configured for each component: CPU usage, memory usage, thread usage, and message frequency. The properties of the components are defined as a range around the parameterized resource usage. In order to average the results, we log the resource usage of the monitoring each second, resulting in 180 independent measurements for each monitoring configuration and system size. To ground this results, we specify the applied computer in our simulation. We used a X201 Tablet from Lenovo, with an Intel Core i7 L 620 and 4 GB of memory.

Table 1 Skill prioritization of the scenario

	Drive	Grab	Detect	Localize	Control
Task-related:	–	–	–	–	–
Search task	7	–	7	4	8
Collect task	1	7	4	–	8
Transport task	7	–	–	7	8
Dump task	–	7	–	–	8
Pessimistic	9	9	9	9	9
Individual	7	7	7	7	8
Operational	7	7	7	7	8
Optimistic	4	4	4	4	4

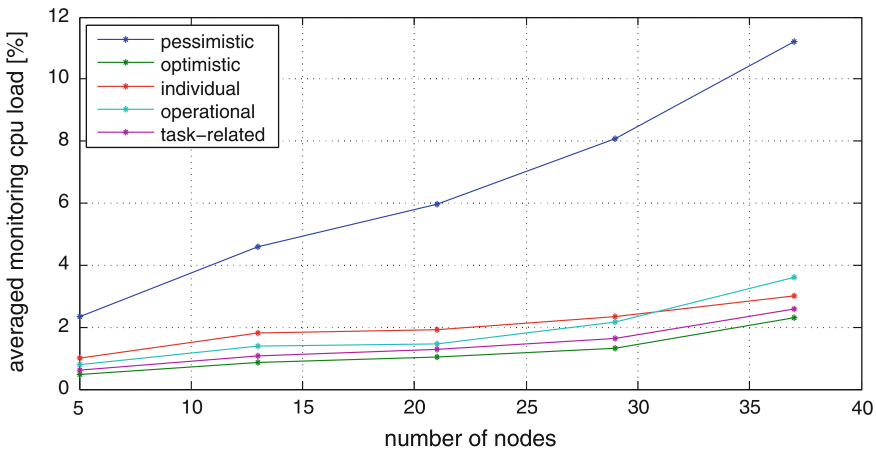

Fig. 4 Scalability of the adaptive meta data-based monitoring

5.3 Discussion of the Results

We present the combined efficiency and scalability qualities of the performed simulation in Fig. 4. While the scalability is represented as the degree of the rise in the curve, the efficiency property is depicted as the ratio between the pessimistic and the instances of the proposed monitoring. As seen in Fig. 4, the individual configuration reduces the monitoring overhead in respect to the pessimistic one by 60.17 %. Furthermore, the operational adaptation provides a gain of 69.98 %, while an additional adaptation in respect to the tasks yield to 76.94 %. This is 4.46 % above the optimistic monitoring. These efficiency gains are determined in respect to a system size of 13 components. In contrast to the optimistic monitoring, the acquisition progress (observation frequency) can be intensified almost to the level of the pessimistic monitoring if required. However, the initial high gain of the individual configuration results from high resource overhead differences in high priority values. For example, a constant and static monitoring with a priority of 8 already decreases the resource overhead to 50.54 %. We introduce our search-and-collect scenario as a vicarious example for a single robot mission. In general, the efficiency gain depends on the specific application context and the operational situation. Therefore, the presented results should not be seen as a general proof-of-concept, but provide indication of resource gains to expect.

Additional, we have gathered empirical results from a real-world setup. The proposed monitoring was implemented on a Mid-Size-League RoboCup robot [16], where the underlying PC is an Intel Core i7 M 620 with 4 GB memory. The robot system consists of six components to observe. In this setup, the system monitoring yields to approximated 1.65 % of resource usage (averaged CPU load).

6 Conclusions

Monitoring faces special challenges in the robot domain. While failure recovery relies on up to date and extensive system observations, the resource overhead of the monitoring process should be minimal. In order to contribute to this challenge, we propose a model-based and adaptive monitoring. The key assets of the proposed approach are the individual configurability and the adaptation in respect to runtime variations and robot task changes. Besides active monitoring support from system components, our approach focuses on the observation of generic, external state information, like resource usage, or communication properties (passive monitoring). The configurability is based on a hierarchical layered robot model for individual specification of the relevance and monitoring scope of components. Moreover, we propose an adaption of the individual monitoring in respect to observation deviations from typical values. High deviations are taken as indication of conspicuous system behavior resulting in an intensified monitoring, while low deviations reduce the monitoring efforts and contribute to lower resource usage. Performing a robot mission often requires only subparts of the robot skills for the current task. Therefore, we further extend our proposed monitoring concept to adapt the focus to the current task. This limits the monitoring scope to the currently required skills. Hence, we claim that the required resource usage is significant reduced, while the quality of system observation is maintained. In order to support this statement, we evaluate the proposed monitoring concepts against a constant and static monitoring realization often found in practice. As a result, the proposed model-based, adaptive monitoring provides significantly reduced resource costs of 76.94 %, while providing the capability of intensified observation in suspicious situations.

The individual specification of the monitoring scope and the characteristic value ranges relies on expert knowledge of the system and domain. In order to reduce this manual specification effort, our next goals include the usage of online machine learning techniques to automatically learn these settings.

References

1. Lee, S., Cho, H., Yoon, K.J., Lee, J.: Intelligent Autonomous Systems 12. Springer Advances in Intelligent Systems and Computing, Vol. 194 (2013)
2. Jung, M., Kazanzides, P.: Run-time Safety Framework for Component-based Medical Robots. In: International Conference on Cyper-Physical System, Philadelphia (2013)
3. Hill, J., Sutherland, H., Staudinger, P., Silveria, T., Schmidt, D.C., Slaby, J., Visnevski, N.: OASIS : An Architecture for Dynamic Instrumentation of Enterprise Distributed Real-time and Embedded Systems. Computer System Science and Engineering (Real-time Systems) (2011)
4. Steinbauer, G., Wotawa, F.: Detecting and locating faults in the control software of autonomous mobile robots. In: International Joint Conference on Artificial Intelligence, Edinburgh (2005)
5. Dekhil, M., Henderson, T.C.: Instrumented Sensor System Architecture. The International Journal of Robotics Research 17(4) (April 1998) 402–417.
6. Elkady, A., Sobh, T.: Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. Journal of Robotics (2012)
7. Quigley, M., Conley, K., Brian P., G., Josh, F., Tully, F., Jeremy, L., Rob, W., Andrew Y., N.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software. (2009)
8. Volpe, R., Nesnas, I., Estlin, T., Mutz, D.: CLARAty: Coupled layer architecture for robotic autonomy. JPL Technical Report (December) (2000) 116
9. Simmons, R., Apfelbaum, D.: A task description language for robot control. International Conference on Intelligent Robots and Systems **3** (1998)
10. Steinbauer, G., Martin, M., Wotawa, F.: Real-Time Diagnosis and Repair of Faults of Robot Control Software. In: RoboCup 2005: Robot Soccer World Cup IX, Springer (2006) 13–23
11. Grosclaude, I.: Model-based monitoring of component-based software systems. In: International Workshop on Priciples of Diagnosis, Carcassonne, France (2004)
12. Kirchner, D., Niemczyk, S., Geihs, K.: RoSHA : A Multi-Robot Self-Healing Architecture. In: RoboCup2013: Robot World Cup XVII, Eindhoven, Springer (2013)
13. Avizienis, A., Laprie, J.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1) (2004)
14. Kirchner, D., Geihs, K.: Qualitative Bayesian Failure Diagnosis for Robot Systems. In: Submitted to International Conference on Intelligent Robots and Systems, Chicago, IEEE (2014)
15. Kirchner, D., Saur, D.: Reliable Robotics - Diagnostics++. In: ROS Developer Conference, Stuttgart (2013)
16. Haque, T., Geihs, K., Kirchner, D., Opfer, S., Saur, D., Witsch, A.: Team description: Carpe noctem 2013. Technical report (2013)