Model-Based Reasoning for Self-Adaptive Systems – Theory and Practice

Gerald Steinbauer and Franz Wotawa[⋆]

Technische Universität Graz, Institute for Software Technology,
Inffeldgasse 16b/2, A-8010 Graz, Austria
{gstein,wotawa}@ist.tugraz.at,
http://www.ist.tugraz.at/

Abstract. Internal faults but also external events, or misinterpretations of sensor inputs as well as failing actuator actions make developing dependable systems a demanding task. This holds especially in the case where systems heavily interact with their environment. Even in case that the most common faults can be handled, it is very unlikely to capture all possible faults or interaction patterns at development time. As a consequence self-adaptive systems that respond to certain unexpected actions and observations at runtime are required. A pre-requisite for such system behavior is that the system itself has knowledge about itself and its objectives, which can be used for adapting its behavior autonomously. In order to provide a methodology for such systems we propose the use of model-based reasoning as foundation for adaptive systems. Besides lying out the basic principles, which allow for assurance of correctness and completeness of the reasoning results with respect to the underlying system model, we show how these techniques can be used to build self-adaptive mobile robots. In particular the proposed methodology relies on model-based diagnosis in combination with planning. We also discuss modeling issues and show how modeling paradigms influences the outcome of model-based reasoning. Moreover, we introduce some case studies of self-adaptive systems that rely on model-based reasoning concepts in order to show their applicability in practice. The case studies include mobile robots that react on hardware and software failures by applying corrective actions like restarting subsystems or reconfiguration of system parameters.

1 Introduction

What makes humans successful in interacting with other humans and nature and allows for reacting to situations, which have not been faced before? There are many potential answers like the ability to learn, to make associations, and to reason using the available knowledge that has been gained during a lifespan. Especially, reacting appropriately to new conditions is important and allows for exploring new territories. Such a capability would also be strongly desired for autonomous systems like mobile robots and are necessary if we want robots to be of more general use for us. A household robot for example, has to adapt to new homes and changes in the environment smoothly. This kind of self-adaption is not the only one we are interested in.

[⋆] Authors are listed in alphabetical order.

Consider for example a robot where a fault in the hardware occurs. For example, there is a problem with a connector causing the software to crash. Even in case the software is automatically restarted, the fault in the connector again leads to a crash. Without having control on possible repair actions allowing us to overcome such difficulties the robot would be lost. A similar situation would happen when a sensor delivers wrong values or an actuator that does not work as expected but the control system is not capable to deal with such faults and their consequences. It is worth noting that handling all possible faults in all situations explicitly via introducing hardcoded measures seems to be impossible or at least a very tedious task during design. Hence, again there is a strong requirement for self-adaption in the field of autonomous systems.

The main objective of the work described in the paper is to make an autonomous mobile system more robust and flexible. Therefore, all sources of malfunctioning, i.e., faults in the hardware or software, not considered interactions with the environment at design time, and wrong beliefs (which might originate from sensor data interpretations) have to be handled. Flexibility can be gained by allowing a system to continue its mission even in case of faults if they for example belong to system parts that are not important to fulfill a give mission. Alternatively, the system reconfigures itself in order to still provide the necessary functionality.

In order to achieve all of the mentioned objectives for enabling real autonomy the robot control system has to have means for anomaly detection, root cause analysis, and repair, which can be summarized as diagnosis. In anomaly detection the expected behavior in a current state is compared with the observed behavior. In model-based reasoning the expected behavior we obtain from a formal description of the system and its environment, i.e., the model. We assume that the models for behaviors are correct, i.e., they reflect the real actual behavior of the robot system. The observations in the robotics domain come from the sensors. Root cause detection is the part of diagnosis where the reason for a deviation between the expected and the observed behavior is identified. We make use of the fault detection model for root cause analysis. The last step of diagnosis is repair where appropriate measures for eliminating the undesired effects of a root cause are taken. It is worth mentioning that repair does not necessarily mean to replace a faulty component with a spare part. Repair can also be to take compensatory actions for eliminating undesired effects or reconfigurations of a system for enabling the wanted functionality.

So why is diagnosis of autonomous systems so difficult? The main reason lies in the increase of system influences that cannot be foreseen anymore. Almost all nowadays systems are designed for a particular purpose taking care of the design constraints, which are more or less limited by the purpose and the field of application. Most of the constraints are known and careful engineering allows for ensuring functionality as expected over lifetime. Of course maintenance work is necessary but due to knowledge about the system and its comprising parts even maintenance actives can be planned and managed. The situation changes completely when increasing autonomy or the field of application. In both cases the number of design constraints increases as well. As a consequence there is a high likelihood that interactions between the system and its environment, which are not considered during engineering, take place. Such interactions potentially lead to undesired behavior or even a failure.

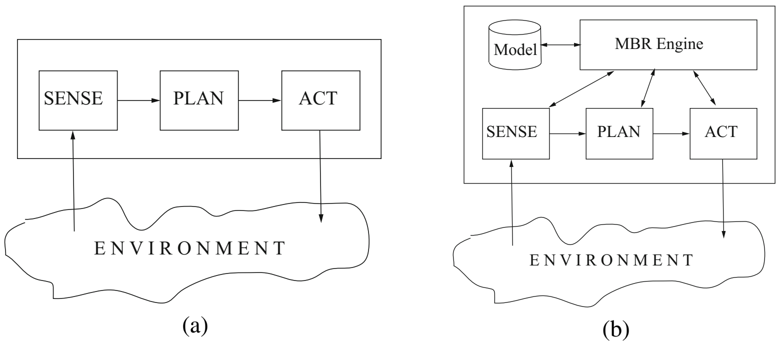


Fig. 1. The sense-plan-act control architecture (a) and its adaptive variant (b)

When it is not possible to keep all necessary design constraints in mind, we have at least to think about alternatives. In this paper we suggest to move some of the hard coded decisions a system can take to a more flexible control system that relies on a model of the system as well as its environment. We argue that unforeseen interactions with the model-enhanced system might have not the same negative impacts. The reason lies in the fact that the model describes the functionality, the behavior, and other prerequisites in a declarative way. The model does not describe how to solve a certain problem. Hence, in case of a misbehavior possible solutions can be derived from the model directly using a declarative reasoning engine. Of course the reactions to events that are not considered might be limited if the models do not reflect the behavior of the overall system.

In this paper we discuss model-based reasoning and its application to self-adaptive systems in the context of autonomous mobile robots. We extend the standard sense-plan-act control paradigm depicted in Figure 1(a) with a model-based reasoning engine (Fig. 1(b)). In the original control paradigm actions are selected on bases of the current state, which is determined by the measurements obtained from the sensor and the internal belief, and the mission (or goal) of the robot. The action selection is done by the planning module and the action module executes the actions. As a consequence of action execution the environment is changed, e.g., by driving from a position to another, which is again measured using the sensors. Hence, there is always a feedback between action execution and sensing actions in robot control. Although, there are many other control paradigms for robots available, e.g., Rodney Brooks's subsumption architecture [1], the underlying principle of actions selected because of sensor information and internal belief remains the same.

Given the robot's control loop we are able to identify three kind of faults that can arise during a mission. First, the sensors might deliver wrong values. Wrong in this context mean that they do not reflect the state of the environment. For example, a robot playing soccer sees the ball at a position but the ball is not there. Second, the execution of actions might fail but remains undetected. An example for this fault is that a robot want to release and object but fails and still carries the object with her. These two kind of faults have as consequence a faulty internal belief, which might lead to wrong plans and dreadful control decisions causing a mission to fail. The third fault category that is relevant are faults in the hardware or software. Note that we do not require a design error causing later on a fault and a failing behavior of the robot. Due to interactions with the environment, e.g., when being hit by another entity, the robot hardware might be affected.

The extended control paradigm given in Fig. 1(b) adds a model-based reasoning (MBR) engine. The MBR engine makes use of a model of the system in order to correct the execution in case of a fault. The model is assumed to capture the structure of the system and the behavior of the components. Moreover, system properties as well as environmental constraints should be part of the model in order to detect situations that are impossible in reality. The same model is used for detecting inconsistencies as well as for fault localization and repair. For the latter the knowledge about functionality of system parts, the mission, and repair actions have to be added. In the following section we introduce model-based reasoning and how diagnosis can be integrated into the planning process.

There are many challenges to be solved when using MBR for the purpose of self-adaptive systems. First of all, developing a model is a hard task. The model should be compositional, declarative, and capture the function and behavior of the system in a depth allowing to use an MBR engine for fault detection, localization, and repair. However, if the model is implemented using basic design principles like the *no function in structure* principle, and *compositionally* large parts of the model can be re-used in other systems. The second challenge is due to underlying reasoning mechanism, which is based on non-monotonic reasoning. The used reasoning technique allows for stating hypotheses about the health state of components. Such hypotheses are asserted and retracted for computing root causes. The computation complexity of underlying algorithm is high in the worst case, and counter measures like the use of heuristics or certain optimality criteria have to be undertaken. Note that in practice diagnosis of systems comprising hundreds to several thousands of components is possible in a reasonable time. Using integrated circuit examples from industry (ISCAS benchmark suite) it had be shown that diagnosis of single faults in circuits with up to 1000 components can be calculated in less than 200 milliseconds on a standard computer while diagnosis of triple faults can run for up to 5 minutes [2]. If real-time requirements have to be considered, e.g., react on signals within a few milliseconds, special domain-dependent encoding of the diagnosis problem can be used [3].

In summary we provide the following contributions:

– We discuss an extension to the well-known sense-control-act architecture used in autonomous systems. In particular, we add an explicit component for handling faults, external events, and misinterpretation of sensor inputs.
– We formally introduce the underlying ideas and basic concepts of model-based reasoning and illustrate them using an example from the robotics domain.
– Furthermore, we introduce an algorithm for diagnosis and self-adaptation. The self-adaptation algorithm combines diagnosis and planning in order to implement the extended sense-plan-act control architecture.

The rest of this paper is organized as follows. In the next section we discuss research related to MBR and self-adaptive systems. We continue with a formal introduction to

MBR. There we explain the underlying definitions using a running example. Moreover, we state a simple algorithm for fault localization and show how this algorithm can be combined with the planning component of the robot's control architecture. In order to show the applicability of MBR to self-adaptive systems, we recall several applications. In particular we show how software can be diagnosed using MBR and how to perform fault localization for particular robot drives. Finally, we conclude the paper.

2 Related Research

The goal to make an autonomous system comprising hardware or software dependable in the sense that the system is able to react to neither modeled nor foreseen circumstances has a long tradition in the areas of Artificial Intelligence, Robotics and Software Engineering.

In Robotics the question how decision making and execution can be robustly organized despite poorly modeled tasks and environments, unreliable perception, and non-deterministic execution arose early. The three-layer-architecture proposed in [4] is basically still the foundation for most robot control architectures. There the authors introduced the idea to structure decision making and execution into three layers with increasing abstraction, deadlines and computational demands. The controller level reacts directly and fast on sensor inputs and is able to cope with local and immediate problems like sensor glitches. The sequencer level is responsible for triggering primitive behaviors in order to achieve a given task. The sequence of behaviors to be executed is generated by the deliberative level. Whereas the deliberate level is able to incorporate persistent changes of the environment or the robot's capabilities in its planning, the sequencer level is able to deal with problems in the outcome of behaviors by issuing a re-planning request to the deliberative level. This approach is basically similar to the proposed method introduced in this paper. However, it differs in two ways. In particular, using a reasoning engine more precise information on the root cause of a problem can be obtained that can be used by the sequencer and deliberative level to achieve more fine-grained modifications of planning and plan execution. For instance, some faulty actions may be excluded from the planning process. Moreover, the reasoning engine is able to deliver information on the proper function of system components, which can be considered by the deliberative level or for reconfiguring the sequencer and controller level. We will present an example for reconfiguration of defective robot hardware later. Such interactions with the system are nor foreseen in the original architecture.

Using the ideas of the three-layer-architecture Kramer and Magee [5] presented a proposal for an architecture for self-adaptive systems. This proposal comprises layers for component control, change management and goal management. The component control is responsible for the reconfiguration of individual components. Such reconfiguration requests are initiated by change management in order to react to states reported by the layer below or to new goals. Action at this layer comprises the creation or termination of components. The goal management is intended to produce plans for the change management layer in order to achieve certain goal or requirements for the overall software system. The main difference of this architecture to the approach presented in the paper is that it deals mainly with the structure of the system. For instance the architecture may take care that a minimum number of redundant modules run at the same time. The architecture does not determine how the software system fulfill its mission. In the approach presented in this paper the planning for achieving a mission and the reaction to problems in the system and its environment are interwoven in one decision making and execution module. Therefore, both parts can benefit from each other. For instance, the mission planner can use information about the current functionality of a hardware component.

In [6] Avizienis and colleagues provided ed definitions for the taxonomy of faults, their sources, their properties, and techniques to cope with them. The taxonomy origins from the secure computing domain and distinguishes basically two type of faults: development and operational faults. The former faults are maybe maliciously introduced in the design and implementation phase whereas the latter faults arise during the operation of the system. The faults mentioned in the introduction the proposed approach is able to cope with (sensing, execution, hardware and software) belong to the former group. We consider development faults only to the extent that they lead to transient problems and can be handled for instance by repeating an action. Even if the taxonomy of [6] considers interaction it is limited to direct interaction of components. In the context of autonomous system a more important source of faults is the interaction with the environment. Moreover, faults in the autonomous decision making system are not considered and posts several new challenges. In [7] an extended taxonomy of faults suitable for autonomous systems was presented. In [8] the taxonomy was enhanced by faults that origin from properties of the used algorithms. Many of the algorithms used today in Robotics are probabilistic methods. Therefore, there is a chance that even the algorithm is implemented correctly it produced inferior results. Moreover, the paper reports a survey of the occurrence of faults and their impacts in real robot systems.

There are many other applications of MBR for adaptive systems. All of the work we are discussing in this section deals with MBR at runtime or at least can be applied to be used at runtime. The first papers deal with on-board diagnosis of cars. Cascio and colleagues [9] as well as Struss and colleagues [10–13] developed systems that are able to find and locate faults of car subsystems during operation. Breaking systems as well as the whole combustion system of cars were modeled. Due to limited computational resources and real-time requirements the authors rely on simplified models that capture the essential parts of the car subsystems but ignoring the real quantitative values. In particular qualitative models [14] are used, where the quantitative values are mapped to qualitative values. For example, instead of considering temperature values like -4 degrees Celsius a qualitative representation like ”*cold*” is used. In their papers the authors demonstrated that the combination of qualitative reasoning models and MBR can be effectively used for on-board fault detection and localization in cars. The repair task, which would follow fault localization, was not handled.

As part of NASA’s Deep Space 1 mission a control system for spacecrafts that has capabilities to adapt itself to faults has been introduced [15, 16]. The underlying idea is to gain autonomy for spacecrafts in order to make increase applicability. The control system comprises a MBR engine that allows for fault localization and re-configuration [17] and a reactive planner [18]. The whole system was implemented and successfully tested during the Deep Space 1 mission. The remote agent experiment is a proof of concept that MBR technology is capable of increasing autonomy and a good bases for self-adaptive systems.

Another domain that is closely related to self-adaptation is reconfiguration where parts of a system are changed to implement a given functionality of the system. This might be changes in the structure of the system or changes in the parameters such that the functionality of the system is adapted to given needs. Configuration and reconfiguration is a still very active research field. Early work that also deals with MBR includes Crow and Rushby [19], and Stumptner and Wotawa [20]. The latter one deals with parameter reconfiguration and uses different component modes to determine valid parameters for the desired functionality. In the paper the authors introduce the foundations for dynamic adaptations of parameters in case of changes in the functionality.

The WS-Diamond project[1] deals with diagnosis and repair of web-services, workflows, and service-oriented architectures. The objective of the project was to achieve self-healing capabilities of web-services [21] where repair is not only a replacement of software components or services but also requires compensating actions for bringing the faulty system again into a correct state. Moreover, the project also focusses on general questions like diagnosability and repairability when using certain models of the system [22]. There are many similarities between WS-Diamond project and our work in self-adaption of autonomous systems. There is a need for combining diagnosis and planning. Moreover, some of the models used for diagnosis of web-services can also be used for software diagnosis of a robot's control program. However, there are also differences that come from the application domain. The environment where web-services are running is more restricted than the real world. In addition faults coming from hardware or the interaction between faulty hardware and software have no counterpart on side of web-services.

Karsai and colleagues [23] introduced a approach for integrating diagnosis and control. Their work is based on Bond graphs for modeling systems in the aeronautic domain. Their work is also based on the underlying ideas of MBD. In [24] the authors present the basic principles using a case study from a fuel system of an airplane. In the domain of model driven development and advanced mechatronic systems the authors of [25] introduced an approach that allows to combine multiple modeling methods to model complex reconfigurable hybrid systems.

3 Model-Based Reasoning

The Artificial Intelligence (AI) field Model-based reasoning (MBR) deals with foundations for obtaining solutions directly from the available knowledge of the real world. The available knowledge, i.e., the model, represents the structure and the behavior of those parts of the world that are necessary to know in the context of the application. The restriction to the context of application is not really a problem because all models like the theory of physics only represent the parts of interest. Using a model directly is in contrast to very early work in AI where knowledge about how to obtain a solution from given facts is used. Hence, in MBR the model does not capture the way of how to

[1] see `http://wsdiamond.di.unito.it/`

obtain solution but states constraints between the model entities. Because of the direct use of model MBR is also called reasoning from first principles.

Using a model directly for obtaining solutions increases flexibility, which is necessary in certain application domains like adaptive systems where the degree of adaption as consequence of events might not be known in advance. It is worth noting that there was a strong need for a flexible method because traditional AI techniques like rule-based systems cause high costs in case of adaptions as part of the usual maintenance process on the available knowledge base. In the early 80th of the last century the foundations of MBR were first published [26–28]. In most of these early papers diagnosis is the underlying application domain. In this paper, we also make use of model-based diagnosis (MBD) and rely on the foundations that goes back to Reiter's [29], and De Kleer and William's [30] work.

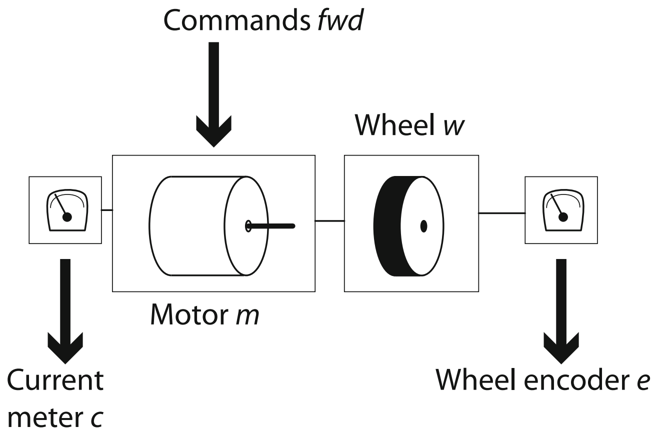


Fig. 2. A partial schematics of a robot drive

Before introducing the formal definitions and an algorithm for MBD, we discuss the underlying idea using a small example from the Robotics domain taken from [31]. Figure 2 depicts the schematics of a part of a robot drive comprising a current meter c, a motor m, a wheel w, and a wheel encoder e. The current meter c is for measuring the current flowing through the motor m. In case of setting m to start moving in a forward direction using the command fwd, there will be a current. Otherwise, we do not measure any current using c. If the motor is running, the wheel w should rotate, which can be measured using the wheel encoder e. Now let us assume that we start the motor using the fwd command, and observe a nominal current flow using c but no information coming from the wheel encoder e indicating a rotating wheel.

Using the given information we are able to identify diagnosis candidates. Because of a measured current flow we can eliminate the cause motor broken when assuming single faults only. A single fault means that there is only on root cause for on or more undesired observations. For instance, if the battery of the robot is faulty several components might show an undesired behavior even the components are correct. Hence, only the wheel or the wheel encoder might be faulty. Of course it can also be the case that an assembly between the wheel or the motor or the encoder might be the reason.

However, in our model we have not introduced the assembly as an own entity. What we can conclude from the example is the following: (1) A diagnosis candidate is a component that when being faulty explains the given observations; (2) We used the behavior of the components and their connections for identifying diagnoses; (3) It is possible to exclude potential causes like the motor from the list of diagnosis candidates; (4) For the purpose of reasoning we used the underlying assumption that a component is correct or that it is faulty. For example, when assuming the wheel w to be broken, there will be no rotation and thus no information coming from the wheel encoder e even in case e is working as expected.

We now formalize the basic idea of MBD. We start with the definition of diagnosis models, which are also called system descriptions (SD) in the MBD context. A diagnosis model comprises the structure of the system and the behavior of components in case the component works as expected and alternatively in case the component is in a particular faulty state. For the case of correctness we use the negation of the predicate AB where AB stands for abnormal. It is worth noting that in case of abnormal behavior, i.e., when AB is true, we do not know the exact behavior. Hence, there are no rules in SD for this case. Alternatively, a component is in a known fault state (or fault mode) F_i. For this purpose there will be a predicate F_i and a set of rules specifying the faulty behavior. For each component $c \in COMP$ we have a set of possible modes where each mode represent a particular state. AB and $\neg AB$ is always part of the possible states. In order to access these states we assume a set $MODES$ that comprises all possible states, and a function $modes : COMP \mapsto 2^{MODES}$ mapping components to the possible modes. For $modes$ we require that $\forall c \in COMP : \{AB, \neg AB\} \subseteq modes(c)$ holds.

In the following we introduce the model using first order logic for our example. Note that the approach is not restricted to first order logic. Any logic that allows for proving satisfiability works.

Motor: In case of a correct behavior a particular motor is running in forward direction if there is command for moving forward cmd_fwd. In this case there is a current flowing through the motor. This current can be of nominal value or higher if there is a strong resistance coming from attached components like wheels. We assume an abstraction function from the quantitative measurement signal to the qualitative observation. Usually continuous signals are noisy and dynamic. Therefore, we use a hysteresis-based approach for the abstraction step [32]. Hence, for all components X that are motors, the following logical sentence formalize their behavior:

$$\forall X : motor(X) \rightarrow \left(\begin{array}{c} \neg AB(X) \rightarrow (cmd\_fwd \rightarrow direction(X, fwd)) \\ direction(X, fwd) \rightarrow torque(X, fwd) \\ direction(X, fwd) \wedge resistance(X) \rightarrow current(X, high) \\ direction(X, fwd) \wedge \neg resistance(X) \rightarrow current(X, nominal) \\ current(X, high) \vee current(X, nominal) \rightarrow direction(X, fwd) \end{array} \right) \tag{1}$$

Wheel: The torque provided via the axle is turned into rotations of the wheel. Only in case of a stuck wheel there is a resistance against the applied torque. But this case is not going to be formalized because we are only interested in the correct behavior. For wheels X we formalize the correct behavior as follows:

$$\begin{array}{l}\forall X : wheel(X) \rightarrow \\ (\neg AB(X) \rightarrow (torque(X, fwd) \rightarrow (rotate(X, fwd) \wedge \neg resistance(X))))\end{array} \tag{2}$$

Current meter: The current meter measures the value of the current. Hence, there is a one to one correspondence of the current flow and the measurement in case of a correct behavior. This behavior of a current meter X can be formalized as follows:

$$\begin{array}{l}\forall X : current\_meter(X) \rightarrow \\ \left(\begin{array}{c}\neg AB(X) \rightarrow (current(X, high) \leftrightarrow observed\_current(X, high)) \\ \neg AB(X) \rightarrow (current(X, nominal) \leftrightarrow observed\_current(X, nominal))\end{array}\right)\end{array} \tag{3}$$

Wheel encoder: The wheel encoder X is providing a frequency only in case of a rotation. For a wheel encoder X we introduce the following logical sentence:

$$\begin{array}{l}\forall X : encoder(X) \rightarrow \\ (\neg AB(X) \rightarrow (rotate(X, fwd) \leftrightarrow frequency(X)))\end{array} \tag{4}$$

What is now missing to complete the model for our example is a description of the structure. We first define the components:

$$motor(m) \wedge wheel(w) \wedge current\_meter(c) \wedge encoder(e) \tag{5}$$

Second, we have to define the connections between the components. This can be done using the following logical sentences:

$$\begin{array}{rcl} torque(m, fwd) & \leftrightarrow & torque(w, fwd) \\ resistance(m) & \leftrightarrow & resistance(w) \\ current(m, nominal) & \leftrightarrow & current(c, nominal) \\ current(m, high) & \leftrightarrow & current(c, high) \\ rotate(w, fwd) & \leftrightarrow & rotate(e, fwd) \end{array} \tag{6}$$

The rules stated in Equations (1) – (6) define the structure and behavior of the model SD of our small example. For this example the set of components comprise 4 elements, i.e., $COMP = \{m, c, w, e\}$. Moreover, the example makes only use of AB and $\neg AB$. Hence, the set of modes only holds these elements: $MODES = \{AB, \neg AB\}$.

We now define a diagnosis problem formally. Although, we borrow the ideas behind model-based diagnosis from Reiter [29], we adapt the definitions in order to deal with fault states. In Reiter's seminal work only the correct behavior of components is used. There are many papers also dealing with fault modes like [33] and [34]. Both papers deal with the relationship between the case of diagnosis without fault modes and the one with only explicit fault modes, i.e., where there is no explicitly given fault behavior. Fault modes usually improve diagnosis, i.e., they help to reduce the number of diagnosis candidates. It is worth noting that sometimes the use of fault modes is not necessary for improving diagnosis. Handling physical impossibilities seems to be sufficient. See [35] for a more detailed discussion on this topic.

Definition 1 (Diagnosis problem). *A diagnosis problem is a tuple $(SD, COMP, MODES, modes, OBS)$, where SD is a model, $COMP$ a set of components, $MODES$ a set of modes, and $modes$ a function mapping components to their modes, and OBS a set of observations.*

The diagnosis problem for the running example from Fig. 2 comprise the system description, the component set, the modes and the $modes$ function, which map each component to the set $\{AB, \neg AB\}$. The observations for this example are: $\{cmd\_fwd, observed\_current(c, nominal), \neg frequency(e)\}$.

A diagnosis in contrast to the original definition of Reiter [29] is a mapping of exactly one mode to each component such that the observations can be explained. We first define the term mode assignment.

Definition 2 (Mode assignment). *Given a diagnosis problem $(SD, COMP, MODES, modes)$. A mode assignment Δ is a set where the following properties hold:*

1. $\forall c \in C : ((\exists m(c) \in \Delta) \wedge m \in modes(c))$
2. $|\Delta| = |COMP|$

Mode assignments are used to specify that a component is in a certain state. Note that for simplicity we do not consider temporal changes in mode assignments. If temporal changes of behavioral states are possible, i.e., in case of non-permanent faults, the definitions have to be extended.

We are now able to define a diagnosis using mode assignments. The underlying idea here is to find a certain state of the system, which comprises the states of each component, such that the corresponding behavior is not in contradiction with the given observation.

Definition 3 (Diagnosis). *Given a diagnosis problem $(SD, COMP, OBS)$. A mode assignment Δ is a diagnosis if and only if $SD \cup OBS \cup \Delta$ is satisfiable.*

For our running example the mode assignment $\{\neg AB(m), \neg AB(c), AB(w), \neg AB(e)\}$ as well as $\{\neg AB(m), \neg AB(c), \neg AB(w), AB(e)\}$ are both a diagnosis.

In practice computing all diagnoses is not required. Instead we are interested in the "most likely" diagnoses or the diagnoses comprising the least faults. In order to define such ranking of diagnoses we make use of some auxiliary definitions. We first define a set $S_m(\Delta)$ for a mode assignment Δ where $m \in MODES$. The set S_m comprises all components in the mode assignment with mode m, i.e., $S_m(\Delta) \equiv_{DEF} \{c | m(c) \in \Delta\}$. We also assume the existence of a function $p_m : COMP \mapsto [0, 1]$ that maps components to its probability of being in state $m \in MODES$. Obviously $\sum_{m \in modes(C)} p_m(C) = 1$ must hold for all components $c \in COMP$ in order to reflect that a component c has to be in exactly one state.

For the definition of minimality with respect to the set S_m of components of a diagnosis that are in mode m we make use of the following thought. A diagnosis should be preferred over another diagnosis if the first one assumes less faulty components. In our case faulty components in a diagnosis have assigned a state m that is not $\neg AB$. There are two possibilities to state minimality using this thought. We first define subset minimality which ensures that there is no diagnosis comprising less correct diagnoses than the given diagnosis.

Definition 4 (Subset minimality). *A diagnosis Δ for a given diagnosis problem is subset minimal if and only if there exists no other diagnosis Δ' for which the following proposition hold: $S_{\neg AB}(\Delta') \supset S_{\neg AB}(\Delta)$.*

Alternatively to subset minimality we might think about preferring diagnoses that are the smallest ones. This definition of minimality is based on the cardinality of sets.

Definition 5 (Minimal cardinality). *A diagnosis Δ for a given diagnosis problem is minimal with respect to cardinality if and only if there exists no other diagnosis Δ' for which the following proposition hold: $|S_{\neg AB}(\Delta')| > |S_{\neg AB}(\Delta)|$*

Minimality with respect to cardinality can be also seen as a global minimality, because there are no diagnoses with less faulty components.

The last definition of minimality is based on the probability of a diagnosis. When assuming that a mode of a component is stochastically independent from each mode of another component, then the probability of a diagnosis Δ is nothing else than the product of the probabilities of all single mode assignments in Δ, i.e., $Prob(x) \equiv_{DEF} \prod_{m(c)\in\Delta} p_m(c)$. The independence assumption is quite reasonable even if component interacts as the probability express the chance that a component fails by itself.

Definition 6 (Most likely diagnosis). *A diagnosis Δ for a given diagnosis problem is the most likely diagnosis if and only if there exists no other diagnosis Δ' for which the following proposition hold: $Prob(\Delta') < Prob(\Delta)$*

All the above minimality definitions are of practical use. In the hardware domain where fault probabilities of components are known a most likely or most probable diagnosis might be the right choice. In situations where the probabilities are not known, minimality with respect to the size of faulty components might be more appropriate. It is also worth noting that probabilities can also be used in the context of model-based diagnosis to determine the next measurements required to distinguish diagnoses. We refer the interested reader to Williams and de Kleer's paper [30] for an introduction to a method for optimal measurement selection with respect to the required number of measurements for locating the fault.

In the following we introduce an algorithm that computes minimal diagnoses up to a predefined number of faulty components. The algorithm starts assuming that all components are working as expected. In case of a contradiction with the given observations, one after the other component is assumed to switch its mode. Again for each of these mode assignments a check of consistency is performed, and so on. The process stops either when the bound is reached or when there are no combinations left. It is also worth noting, that a check is performed whether there exists a small diagnosis where the correct components are a superset of the correct components of the currently computed diagnosis. If this is the case the new diagnosis is removed from the list of candidates, thus ensuring subset minimality.

In Figure 3 we introduce the MBDIAG algorithm. The algorithm assumes that we have a theorem prover that allows for checking consistency. This is done by calling the CHECK method. It is worth noting that the term theorem prover is not only for representing a system for checking satisfiability of a logical formulae. Instead any algorithm

```
Algorithm MBDIAG
Input: A model $(SD, COMP, MODES, modes)$, observations $OBS$, and the number $n_{AB} \geq 0$ of faulty component to be searched for.
Output: A set of diagnoses $R$.

1. Let $R$ be the empty set.
2. Let $DS$ be the set comprising the element $\{\neg AB(c) | c \in COMP\}$, and let $n = 0$.
3. Let $DS'$ be the empty set.
4. For all elements $\Delta \in DS$ do:
   (a) If CHECK$(SD, \Delta, OBS)$ is consistent, then add $\Delta$ to $R$ if there exists no subset minimal $\Delta'$ in $R$.
   (b) Otherwise, for all components $c$ where $\neg AB(c) \in \Delta$ do:
       i. For all $m \in modes(c) \setminus \{\neg AB\}$ add $\Delta'$ to $DS'$ with $\Delta'$ is $\Delta$ where $\neg AB(c)$ is changed to $m(c)$.
5. Let $n = n + 1$ and let $DS$ be $DS'$.
6. If $n \leq n_{AB}$ and $DS \neq \emptyset$, go to 3
7. Otherwise, return $R$ as result.
```

Fig. 3. The model-based diagnosis algorithm MBDIAG

that allows for deciding whether a given system state together with a formalized model is contradicting the given observations or not. Hence, a simulator or a constraint solver can also be used for this purpose.

Obviously MBDIAG terminates because either the bound n_{AB} is reached or no new diagnosis candidates have to be checked. Moreover, MBDIAG has to compute subset minimal diagnoses because the others are not added to the result set R. The worst case complexity is of course exponential in the number of components. However, for single faults the algorithm is feasible requiring $O(|COMP| \cdot |MODES|)$ steps. Even in case of double and triple faults MBDIAG is in the worst case polynomial in the number of components and modes.

There have been many other diagnosis algorithms described in literature. The most closely to ours is Reiter's diagnosis algorithm [29] that was corrected by Greiner et al. [36]. The algorithm is based on conflicts and hitting sets. A conflict basically is a set of components that when assuming to behave correctly reveal an inconsistency. By combining all these conflicts (via the computation of hitting sets) all minimal diagnoses can be obtained. Another approach that is based on an assumption based truth maintenance system [37] was introduced by de Kleer and Williams [30]. Other papers introducing algorithms for model-based diagnosis include [38], [39], [40], and [41]. The last three publications are of particular interest because they make use of constraints for representing models and observations.

3.1 Smart Control

What is missing after performing fault detection and localization is repair. There are many ways to bring a system to a correct state after a failure. The most straightforward way is replacing broken components with their spare parts. This cannot be done in all application domains. Consider for example a spacecraft on mission. Replacing a component there is obviously not a feasible option. Repair can also be performed via changing the system's structure or configuration in order to meet the current needs. In this case broken components are excluded and other components take their place. In practice usually for this case redundant components are used. However, it is also often possible to use implicit redundancy within an application to make a replacement that allows for still reaching mission goals. For example, if communication via WiFi is not possible any more, maybe it is possible to communicate via another means for communication, which might not be available always but at least sometimes. Hence, performance of the overall system will degrade but not to a point where the mission is in danger.

For coming up with a smart repair engine knowledge of possible repair actions as well as the provided functionality and the goal has to be formalized. Moreover, the results of diagnosis have also to be considered. We do not discuss planning and the planning problem in detail here and refer the interested reader to other publications [42, 43]. Instead we discuss smart plan execution, which is required for really smart control. Combining MBD and planning is not new. Sun and Weld [44] introduced the use of planning for diagnosis with the purpose of controlling the diagnostic process itself. Friedrich and Nejdl [45–47] formalized the process of diagnosis and repair, and also distinguishes (repair) actions from observations. Our work is based on the previous work and focusses more on the execution of control using the sense-plan-act paradigm (see Fig. 1) of mobile autonomous robots.

In the following we introduce the algorithm EXT_PSA_ARCH [31] that is depicted in Figure 4. The algorithm makes use of a function SENSE, which returns the current state of the autonomous system comprising the internal state and the information obtained from the sensors. We assume that only reliable sensor information is given. Therefore, if a diagnosis indicates a sensor fault, the particular sensor is ignored and its information is not provided anymore. Although time is not handled explicitly in the algorithm it is worth noting that time exceeds during execution. Thus `SENSE` represents the state at a discrete point in time only, where measurements and the internal state are observed.

The control algorithm starts computing a plan based on the available information, i.e., the current state provided via the SENSE function, and the planning knowledge based M_p together with the goal state S_G. At the beginning all actions that can be performed by the system are functioning, and are therefore available for planning. Afterwards, the plan is executed by sequentially executing the actions. First, the pre-conditions of the current action a are checked. In case that the pre-conditions are not fulfilled in the current state, the action cannot be performed. This can happen due to an external event. As a consequence, re-planning has to be performed and plan execution starts again using the new plan.

If the pre-conditions of an action are fulfilled, the action is executed. This execution might be terminated returning a failure. In this case diagnosis has to be performed that returns diagnoses using the MBDIAG method, from which a leading diagnosis is obtained. A leading diagnosis can be either the smallest diagnosis or the one with the highest fault probability. For simplicity we do not handle the case of multiple diagnoses that cannot be distinguished with the available information. If such case occurs,

```
Algorithm EXT_PSA_ARCH
Input: A planning model $M_p$, a diagnosis model $(SD, COMP, MODES, modes)$, and the
goal state $S_G$.
Output: Computes and executes a plan from the current state to $S_G$.

1. Let $p$ := PLAN($M_p$,SENSE(),$S_G$).
2. While $p$ is not empty do:
   (a) Let $a$ be the first action of plan $p$.
   (b) Remove $a$ from $p$.
   (c) If the pre conditions of $a$ are not fulfilled in SENSE(), then go to 1
   (d) Otherwise, execute $a$.
   (e) If the execution terminates with a failure, let $S_\Delta$ be the result of calling
       MBDIAG($SD, COMP, MODES, modes$, SENSE()), and $\Delta$ be the leading diag-
       nosis of $S_\Delta$.
   (f) $\Delta$ indicates a sensor failure only, then consider $a$ to be executed without failure and
       proceed. Otherwise, remove all actions from the planning model that cannot be longer
       used because of diagnosis $\Delta$. Go to 1.
   (g) If the effects of $a$ are not fulfilled in SENSE() , then go to 1. Otherwise, continue
       executing the plan.
```

Fig. 4. The smart plan execution algorithm EXT_PSA_ARCH

measures for distinguishing diagnoses have to be performed, like providing testing procedures. For example, Wotawa et al. [48] introduce distinguishing test cases for solving such problems. Hence, we assume that we can always determine a leading diagnosis. This leading diagnosis is used to either remove actions that cannot be performed anymore, or to assume that some sensor data is no longer reliable. In the latter case there is no need to apply re-planning. Instead the action is forced to be executed and the procedure continues.

The last possibility for a fault occurring during execution is that the effects of an action are not visible. In this case we again perform re-planning, which might lead to the case where the current action is re-executed again. Note that this might lead to a situation where the robot is executing an action again and again due to a sensor failure or an external event. Therefore, in the implementation the repetition of executions of the same actions should be tracked and handled appropriately. For example, a diagnosis step might also be performed.

4 Example Applications

After introducing the foundations of MBR we discuss two of our applications of MBR in the domain of self-adaptive systems. The applications are in the field of autonomous mobile robots. Their common objective is to provide adaptation in case of faults occurring either in software or hardware. Although both applications share the same methodology, their underlying models are different. In software repair the dependencies between parameters that are passed from one software component to another are used to localize the fault. Whereas in the case of adaptive kinematics control the whole kinematics of a robot drive is modeled, which requires the use of difference or differential equations. Hence, the application also demonstrate very well the bandwidth of models that are used in MBR.

4.1 Software Repair at Runtime

We first discuss the application of MBD to software repair performed during runtime on a mobile robot. The work of Steinbauer et al. [49] we are discussing in this section distinguishes the fault detection from the fault localization part of diagnosis and is therefore different to ordinary MBD where both tasks are usually handled using the same model. The general idea behind the approach is to recover from severe failure as fast as possible. What the authors had in mind was to tackle cases like deadlocks or crashes of software components that would lead to a loss of the mobile robot.

Figure 5 shows the dependencies of the underlying robot control architecture taken from [49]. We see the different components and their potential interactions via messages and events they are sharing. In the following we discuss the approach used for software repair comprising fault detection, localization, and the repair step, where fault detection starts fault localization, which itself starts repair afterwards.

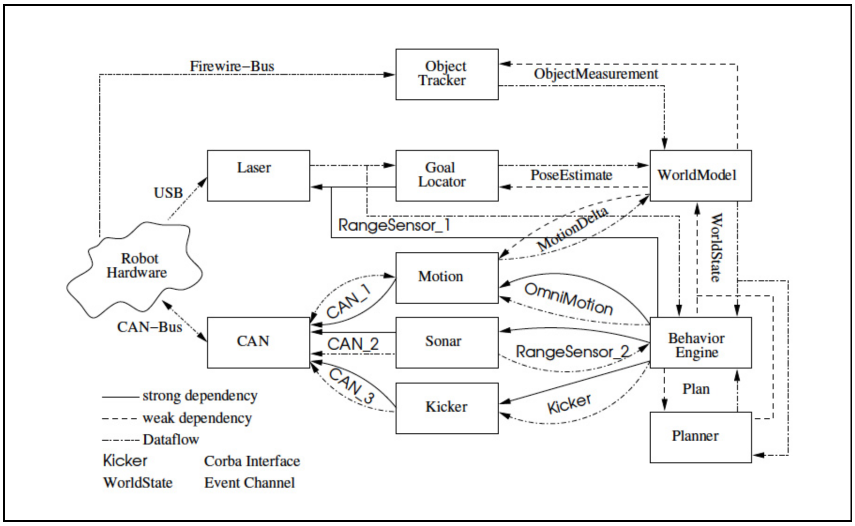


Fig. 5. The robot control architecture from [49]

Fault detection: In [49] fault detection is handled using monitoring. In particular a watch-dog process is invoked, which takes care of messages that are exchanged between the components of the control software, and processes and threads necessary for execution. The watch-dog process checks the following information during runtime:

– Periodic event production, e.g., the motion service has to produce an event every 50 ms.

- Conditional event production, e.g., there has to be an event WorldState produced by the world model component after an event ObjectMeasurement occurs.
- Periodic method calls, e.g., the sonar service of the robot should be regularly called by the behavior engine.
- Spawn processes, e.g., the motion service invokes exactly 6 threads.

The watch-dog raises the fault localization method in case a check fails. In addition every check that fails also has corresponding observations that are given to the fault localization method. The observations state the correctness or incorrectness of certain messages. For example, if the range sensor data is checked to be faulty, a $\neg ok(RangeSensor\_2)$ observation is generated. It is worth mentioning that the overhead for running the monitoring process can be neglected. In [49] the authors reported less than 1% overhead of CPU time and less than 5 $\%$ of memory consumption.

Fault localization: The fault localization step takes the outcome of detection as input and tries to identify the root cause. Because of runtime requirements and the fact that not the whole behavior of the software system (without replicating the whole program in logic) is available, a simplified model was used in [49]. The model itself only takes care of dependence information. A component has a dependence relation with another component if there is a message flow. The underlying idea is not new in the software domain. Friedrich et al. [50] used data and control dependences for locating bugs in VHDL programs. Later Wotawa [51] proved the equivalence of Mark Weiser's slicing [52] and the dependency-based model.

The idea of the dependence model is as follows. Every component that is correct, should produce a correct output in case the inputs are correct too. We only need to formalize this idea where *ok* ($\neg$*ok*) is used to represent a correct (incorrect) value of either an input or output. We describe the model behind the fault localization step using a small example. Consider the software architecture from Figure 5 and focus on the components CAN, Motion, and Sonar only. Between CAN and Motion messages are send on connection CAN_1 and between CAN and Sonar messages are send on connection CAN_2. Since we are only interested in software faults, no information of the hardware is given, and therefore the CAN component has no input in our model. Hence, we finally obtain the following system description SD for the small subsystem:

$$\begin{array}{l}\neg AB(CAN) \rightarrow ok(CAN\_1)\\ \neg AB(CAN) \rightarrow ok(CAN\_2)\\ \neg AB(Motion) \wedge ok(CAN\_1) \rightarrow ok(MotionDelta)\\ \neg AB(Sonar) \wedge ok(CAN\_2) \rightarrow ok(RangeSensor\_2)\end{array}$$

In the model we have three components $COMP = \{CAN, Motion, Sonar\}$, each having assigned two modes AB and $\neg AB$. Let us now assume that the fault detector checks fail for the signals MotionDelta and RangeSensor_2. In this case we have the following set of observations:

$$OBS = \{\neg ok(MotionDelta), \neg ok(RangeSensor\_2)\}$$

Obviously when assuming all components to be correct, we obtain a contradiction. It is easy to show that only the following two mode assignments are subset minimal diagnoses:

$$\{AB(CAN), \neg AB(Motion), \neg AB(Sonar)\}$$
$$\{\neg AB(CAN), AB(Motion), AB(Sonar)\}$$

When considering the smallest root cause first, only the diagnosis indicating that the CAN module is faulty remains.

Repair: After localizing the fault Steinbauer et al. introduced a repair approach where the restart faulty components. In some cases it is also necessary to restart components that are connected with the faulty ones. The information whether this is necessary is stored in the strong dependencies (see Fig. 5). If a component like Can fails also the Sonar module has to be restarted. For the example given in the discussion of fault localization we see that only CAN has to be restarted in order to bring the system into a correct state.

The repair step ensures that components, which are relevant for a certain fault, have to be restarted. This reduces time for bringing a system back to operations. Otherwise, the whole system has to be restarted, which usually takes a much longer time.

Steinbauer et al. also give some initial results in their paper. In particular they consider two situations. In one the Laser Service was killed. In the other the World Model was externally terminated. In both cases the diagnostic process detected the failure, localized the root cause, and brought the robot again to a state where ordinary operation was guaranteed. Note that the authors also reported the use of the system during a RoboCup tournament where a crash related to the image processing module has been solved without human interaction.

From the discussed application on software repair at runtime we are able to draw the following conclusions: (1) The approach allows for effectively autonomous repair of software in case of severe faults like crashes and deadlocks. (2) The use of simple models like the dependency-based model are sufficient in this application domain. (3) The MBD approach can also be used in cases where fault detection via monitoring is separated from fault localization. In this case the monitoring component has to deliver the formal observations. It might be necessary to perform an abstraction step for this purpose where continuous values are mapped to a qualitative value like ok. Note that there are other approaches to software debugging where the program itself is used directly as a model. See for example [53–55]. These approaches can hardly be used directly for runtime diagnosis and repair because of computation requirements. The availability of the proper models is a crucial issue for the acceptance of MBR approaches. For many application the model are handcrafted which is not feasible for larger system or if one do not has access to all internal details of a module. In [56] the authors presented a machine learning approach for model acquisition in the context of robot control software.

4.2 Adaptive Kinematics Control

The second application we discuss is a model-based framework that is able to deal with hardware faults in the driving unit of an autonomous mobile robot. The work presented in [57] focusses on the issue of retaining as much as possible of the robot's functionality when a hardware fault occurs. The proposed framework is self-adaptive in the sense that it follows grateful degradation once a fault is detected. If the system is able to compensate a detected fault it uses model-based reconfiguration. If the system is not able to compensate the fault it reduces the functionality to a lower level that is known and stable. Hence, the knowledge of the degraded system can be used in planning and control afterwards.

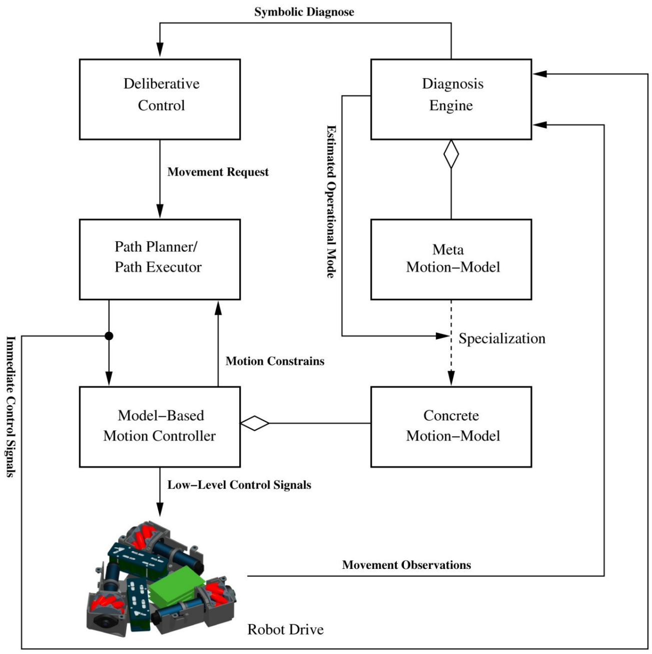


Fig. 6. A adaptive robot control framework

Figure 6 depicts the proposed adaptive robot control framework. The left side of the figure is a classical navigation stack for an autonomous robot [58]. A deliberative control module such as an abstract planner chooses a location where to go next. Such a movement request is communicated down to a path planner, which calculates a collision-free path to the requested goal. Once a feasible path is generated the path executor guides the robot along that path by sending immediate motion commands to the motion controller. The controller converts the motion commands in low-level control commands for the motors, e.g., speed and direction. Such control architectures (abstract control flow) and controllers (dedicated control laws) are usually designed and optimized off-line and do not change during operation. In particular the controller sticks on the kinematics of the robot, the mapping of the control parameter to the robot's motion.

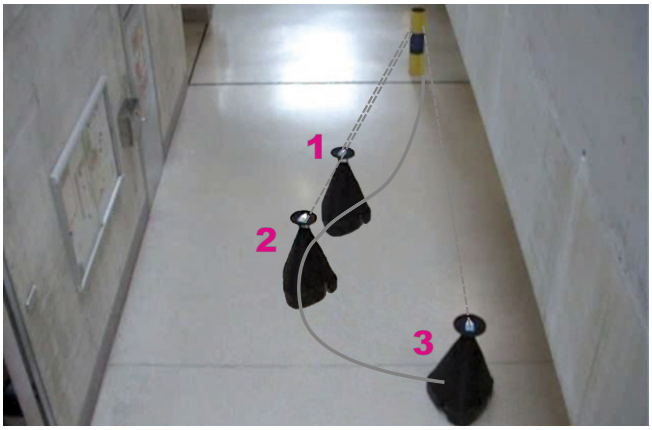
In the case of a hardware fault, e.g., a motor is stuck, the kinematic changes drastically and immediately. If the control architecture is not aware of such changes, it continues its operation and may endanger the mission, the robot, and its environment. This happens because of the changed and now unknown behavior of the system.

In order to allow the robot to react actively to such incidences the control architecture is extended with diagnosis and reconfiguration modules shown on the right side of Figure 6. The basic idea is that the architecture determines the operational mode of the robot drive, i.e., one or more nominal modes and a number of faulty modes, and adapts the model-based controller on-the-fly to reflect the new kinematics that actually rules the robot. As long as the model-based controller is able to adapt to the new kinematics, it simple transparently remaps the commands accordingly. In this case there is no need that the upper modules recognize that there is a fault. If the controller is not able to fully compensate a fault, it degrade the functionality to a lower but known and stable level. Moreover, it informs the upper modules about the degradation in order to allow the path-planner and the deliberative control to adapt their plans to the new situation.

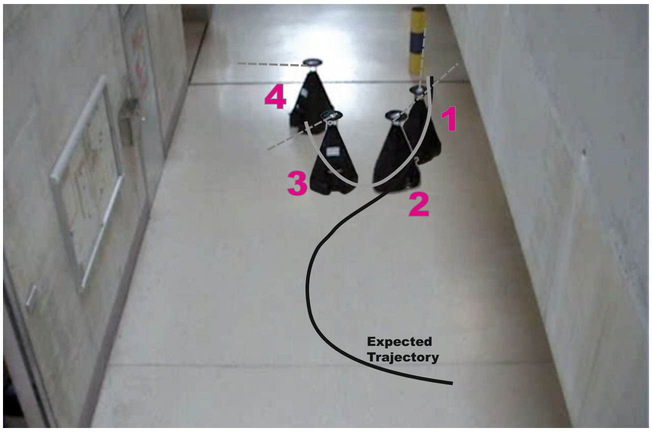
Figure 7 shows a practical example how the adaptive kinematics control works. The example shows an omni-directional autonomous robot executing the so-called *Eye-catcher Task*. For this task the robot has to follow some given path while always looking towards a fixed point (the colored pole on the figures). This task requires omni-directional motion capabilities. The robot has to control all three degrees-of-freedom (DOF) on the plane. Sub-figure (a) shows the correct behavior when the robot works as expected. The robot drives along a s-shape trajectory. The robot is always directed towards the pole (see time steps 1,2 and 3). Sub-figure (b) depicts the faulty behavior when one of the three motors of the robot fail. The fault occurred between time step 1 (still correct) and time step 2 (already a misalignment in the orientation). The control architecture continues its control without awareness of the fault. This leads to an unexpected behavior (time step 3 and 4). Sub-figure (c) shows the robot behavior using the self-adaptive control architecture. The fault was detected but the model-based controller was not able to retain the full functionality. The robot possesses three motors which is the minimum number to provide omnidirectional motion. If one fails the robot has to degrade its functionality. That is exactly what the model-based controller does. It gives up one DOF (direction) and retains the capacities that allows it to follow the trajectory. It is clearly shown that the robot follows the trajectory all the time (time steps 1-4). Only the direction towards the pole is not maintained anymore. Moreover, the path-planner and path-execution engine is informed about the limited maneuverability and adapts the immediate control signals accordingly.

We now discuss the underlying fault detection and localization process as well as the reconfiguration part.

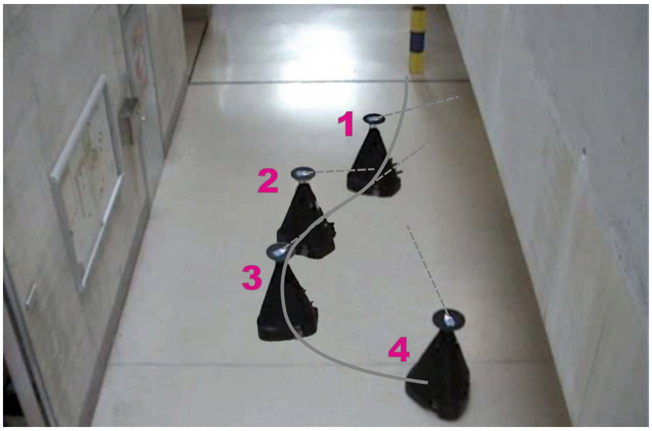
Fault Detection and Localization: Due to the fact that a robot drive is a dynamic system, the actuators and sensors are affected by noise, and the control system has to distinguish several operation modes (different nominal and faulty modes) the fault detection and localization follows the following approach. The behavior of the system is modeled using a hybrid automaton [59]. Figure 8 shows a hybrid automaton for the example part of robot drive depicted in Figure 2. It comprises continuous states, discrete mode states, continuous behaviors for each mode, mode transitions, and probabilities for each transition. The example comprises the discrete modes m_1 (nominal), m_2 (faulty - no torque), m_3 (faulty - stuck), and m_0 (unknown mode). The example transition probability $P_{1,2}$



(a) Correct behavior



(b) Faulty behavior



(c) With self healing

Fig. 7. Adaptive kinematics control in action

is the probability of a change from nominal mode m_1 to faulty mode m_2. The example shows a fully connected automaton. Note that some of the connections can be missing if the transition probability is zero. For instance the probability of a transition from faulty mode m_3 to unknown mode m_0 is quite unlikely. For each discrete mode a continuous behavioral model that relates the continuous input and output values of the system is attached. The continuous input is the voltage applied to the motor u_m. The continuous outputs are the measured rotation velocity ω_e and the current drawn i_c. The continuous behaviors are expressed by differential equations. The equations for mode m_1, m_2 and m_3 are shown in Figure 8. The matrix A comprises constants related to the motor such as its mechanical and electrical properties. The term W_i represents a normal distributed random variable representing the noise in the continuous behavior of mode m_i. Please note that the equation for mode m_3 is no differential equation but simply constraints the rotational velocity to zero. As the unknown mode m_0 comprises no information on the actual behavior no equation is given at all. This does not restrict the variables in any way.

$$\frac{t}{dt}\begin{pmatrix} i_c \\ \omega_e \end{pmatrix} = A\begin{pmatrix} i_c \\ \omega_e \end{pmatrix} + bu_m + W_1; m_1 \tag{7}$$

$$\frac{t}{dt}\begin{pmatrix} i_c \\ \omega_e \end{pmatrix} = A\begin{pmatrix} i_c \\ \omega_e \end{pmatrix} + W_2; m_2 \tag{8}$$

$$\omega_e = 0 + W_3; m_3 \tag{9}$$

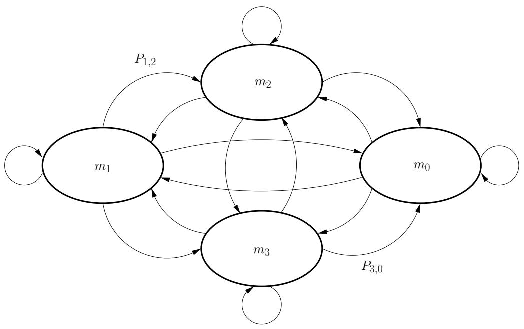


Fig. 8. Hybrid automaton for the example part of a robot drive. The upper part shows the differential equations for the behaviors in the nominal, no torque and stuck mode. The lower part shows the possible mode transitions.

Fault detection and localization is done by estimating the continuous state using the different behavioral models and a multi-hypotheses tracking approach to detect the current discrete mode. The goal is to find the most probable mode the system is in based on the past values of the continuous input and outputs. State estimation for the continuous states is necessary because the system and the measurements are noisy. The multi-hypotheses tracking returns that sequence of discrete modes (changes) that best fit with the continuous state estimation.

If the most probable mode is a faulty mode a fault is detected. The most probable mode gives also information about the fault localization as each fault mode is connected to some fault scenario. It is worth noting that one has to provide a discrete mode, transition probabilities, and a continuous behavioral model for each fault to be detect using the hybrid automaton approach. Faults that are not modeled are caught by an unspecific behavioral model for mode m_0.

Note that the approach follows our given basic definitions of diagnosis using fault modes. The only real difference is that a automaton is used instead of a logical description of the behavior. A mode assignment or in this case a particular state, is a diagnosis if the output is equivalent with the expectations.

Model-based Reconfiguration: Once the most probable operation mode has been identified the adaptive controller architecture starts repair using reconfiguration.The identified state gives information about the faulty components and their behavior, e.g., Motor 3 is stuck. The reconfiguration is achieved by treating the faulty components as additional kinematic constraints. A blocked motor cannot be controlled any more and does not provide any motion. Using this information one is able to determine the actual kinematics of the robot on-line. Moreover, using the actual discrete mode one can derive degradations in capabilities. Like in the *eye-catcher* example above. Because of the geometry of the robot drive and the modes of the motors one can determine whether the robot drive provides full functionality or not. If the functionality degrades (e.g. an omnidirectional robot degrades to a differential robot) this information is communicated to the higher level such as a path planner allowing to reconfigure the path planning to the new kinematics. Even in the case that the fault is very serious and the functionality is completely lost this information is valuable for the higher levels. They may set the robot in a inactive safe state and allow for informing an operator.

5 Conclusions

In the paper we introduced the foundations of model-based reasoning in the context of self-adaptive systems. In particular we present the basic formalization of model-based diagnosis which is founded on earlier work of several colleagues. Moreover, we presented our approach to integrate it into a sense-plan-act architecture of autonomous mobile robots. For this purpose we discussed our extension to a planning and plan execution framework. Moreover, we also discussed two example applications where we used model-based reasoning to increase the robustness of a robot system. In the first application a very abstract dependence model is used to identify failing software components, which are restarted in order to bring the system again into an operational state. The second application makes use of a kinematics knowledge of robot drives. The underlying model captures the behavior using differential equations and explicit fault models. As a result not only faults can be localized but information on degraded behavior can be used to change the control algorithms. There are more applications of model-based reasoning in this domain. We also briefly discussed some of them.

Model-based reasoning provides the necessary foundations for self-adaptive systems. Not only that some of the underlying challenges like diagnosis can be formalized.

The approach also ensures that all results are correct and optimal with respect to the given model. If the model is correct, then the outcome of diagnosis based on the model has to be correct too. Moreover, the approach can not only be used for hardware diagnosis but also for software diagnosis at runtime. The computational overhead of the method is higher but when using simplifying assumptions like that there are only single or double faults, model-based reasoning is feasible. Even larger systems up to several thousand of components can be diagnosed in a reasonable amount of time.

In the domain of autonomous mobile robots using model-based reasoning is a good choice because there the structure of the hardware as well as the behavior of the components are known. Moreover, even fault modes are available for some of the components, which usually make diagnosis more precise. It is worth noting that in cases where no knowledge of the faulty behavior is known, the model-based reasoning approach can still be used. This also distinguishes this approach from others.

Acknowledgments. The work presented in the paper has been funded by the Austrian Science Fund (FWF) under contract number P22690.

References

1. Brooks, R.: Cambrian Intelligence. MIT Press, Cambridge (1999)
2. Pill, I., Quaritsch, T., Wotawa, F.: From Conflicts to Diagnoses: An Empirical Evaluation of Minimal Hitting Set Algorithms. In: Proceedings of the 2nd International Workshop on Principles of Diagnosis (DX 2011), Murnau, Germany (October 2011)
3. Struss, P., Price, C.: Model-based systems in the automotive industry. AI Mag. 24(4), 17–34 (2004)
4. Gat, E.: On three-layer architectures. In: Artificial Intelligence and Mobile Robots. MIT Press (1998)
5. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: 2007 Future of Software Engineering, FOSE 2007, pp. 259–268. IEEE Computer Society, Washington, DC (2007)
6. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (2004)
7. Lussier, B., Lampe, A., Chatila, R., Guiochet, J., Ingrand, F., Killijian, M.O., Powell, D.: Fault Tolerance in Autonomous Systems: How and How Much? In: 4th IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, Nagoya, Japan (2005)
8. Steinbauer, G.: A Survey about Faults of Robots used in RoboCup. In: Chen, X., Stone, P., Sucar, L.E., der Zant, T.V. (eds.) RoboCup-2012: Robot Soccer World Cup XVI. LNCS (LNAI). Springer, Berlin (2013)
9. Cascio, F., Console, L., Guagliumi, M., Osella, M., Panati, A., Sottano, S., Dupré, D.T.: Generating on-board diagnostics of dynamic automotive systems based on qualitative models. AI Communications 12(1/2) (1999)
10. Malik, A., Struss, P., Sachenbacher, M.: Case studies in model-based diagnosis and fault analysis of car-subsystems. In: Proceedings of the European Conference on Artificial Intelligence (ECAI) (1996)
11. Milde, H., Guckenbiehl, T., Malik, A., Neumann, B., Struss, P.: Integrating Model-based Diagnosis Techniques into Current Work Processes – Three Case Studies from the INDIA Project. AI Communications 13 (2000); Special Issue on Industrial Applications of Model-Based Reasoning

12. Sachenbacher, M., Struss, P., Carlén, C.M.: A Prototype for Model-based On-board Diagnosis of Automotive Systems. AI Communications 13 (2000); Special Issue on Industrial Applications of Model-Based Reasoning
13. Picardi, C., Bray, R., Cascio, F., Console, L., Dague, P., Dressler, O., Millet, D., Rehfus, B., Struss, P., Vallée, C.: Idd: Integrating diagnosis in the design of automotive systems. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), Lyon, France, pp. 628–632. IOS Press (2002)
14. Weld, D., de Kleer, J. (eds.): Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann (1989)
15. Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M., Williams, B.: A remote-agent prototype for spacecraft autonomy. In: Proc. of the SPIE Conference on Optical Science, Engineering, and Instrumentation, New Millennium, Bellingham, Waschington, U.S.A. Space Sciencecraft Control and Tracking, Society of Professional Image Engineers (1996)
16. Williams, B.C., Nayak, P.P.: Immobile robots – ai in the new millennium. AI Magazine, 16–35 (1996)
17. Williams, B.C., Nayak, P.P.: A Model-based Approach to Reactive Self-Configuring Systems. In: Proceedings of the Seventh International Workshop on Principles of Diagnosis, pp. 267–274 (1996)
18. Williams, B.C., Nayak, P.P.: A reactive planner for a model-based executive. In: Proceedings 15th International Joint Conf. on Artificial Intelligence, pp. 1178–1185 (1997)
19. Crow, J., Rushby, J.: Model-based reconfiguration: Toward an integration with diagnosis. In: Proceedings AAAI, pp. 836–841. Morgan Kaufmann, Los Angeles (1991)
20. Stumptner, M., Wotawa, F.: Model-based reconfiguration. In: Proceedings Artificial Intelligence in Design, Lisbon, Portugal (1998)
21. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: A soa-based model supporting adaptive web-based applications. In: Proc. of 3rd Conference on Internet and Web Applications and Services (ICIW 2008), pp. 708–713. IEEE, Athens (2008)
22. Cordier, M.O., Pencolé, Y., Travé-Massuyés, L., Vidal, T.: Characterizing and checking self-healability. In: Proc. of 18th European Conference on Artificial Intelligence (ECAI 2008), Patras, Grece (July 2008)
23. Karsai, G., Abdelwahed, S., Biswas, G.: Integrated diagnosis and control for hybrid dynamic systems. In: AIAA Guidance, AIAA Guidance, Navigation and Control Conference, Austin, Texas (August 2003)
24. Narasimhan, S., Biswas, G., Karsai, G., Szemetzy, T., Bowman, T., Kay, M., Keller, K.: Hybrid modeling and diagnosis in the real world: A case study. Working Papers Thirteenth Int Workshop Principles of Diagnosis (June 2002)
25. Giese, H., Henkler, S., Hirsch, M.: A multi-paradigm approach supporting the modular execution of reconfigurable hybrid systems. Simulation 87(9), 775–808 (2011)
26. Davis, R., Shrobe, H., Hamscher, W., Wieckert, K., Shirley, M., Polit, S.: Diagnosis based on structure and function. In: Proceedings AAAI, Pittsburgh, pp. 137–142 (August 1982)
27. Davis, R.: Diagnostic reasoning based on structure and behavior. Artificial Intelligence 24, 347–410 (1984)
28. Davis, R., Hamscher, W.: Model-based reasoning: Troubleshooting. In: Shrobe, H.E. (ed.) Exploring Artificial Intelligence, pp. 297–346. Morgan Kaufmann (1988)
29. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
30. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artificial Intelligence 32(1), 97–130 (1987)

31. Wotawa, F.: Adaptive Autonomous Systems – From the System's Architecture to Testing. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) ISoLA 2011 Workshops 2011. CCIS, vol. 336, pp. 76–90. Springer, Heidelberg (2012)
32. Steinbauer, G., Weber, J., Wotawa, F.: From the real-world to its qualitative representation–practical lessons learned. In: International Workshop on Qualitative Reasoning, pp. 186–191 (2005)
33. de Kleer, J., Mackworth, A.K., Reiter, R.: Characterizing diagnosis and systems. Artificial Intelligence 56 (1992)
34. Console, L., Dupré, D.T., Torasso, P.: On the relationship between abduction and deduction. Journal of Logic and Computation 1(5), 661–690 (1991)
35. Friedrich, G., Gottlob, G., Nejdl, W.: Physical impossibility instead of fault models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 331–336, Boston (August 1990); Also appears in Readings in Model-Based Diagnosis. Morgan Kaufmann (1992)
36. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence 41(1), 79–88 (1989)
37. de Kleer, J.: An assumption-based TMS. Artificial Intelligence 28, 127–162 (1986)
38. Fröhlich, P., Nejdl, W.: A Static Model-Based Engine for Model-Based Reasoning. In: Proceedings 15th International Joint Conf. on Artificial Intelligence, Nagoya, Japan (August 1997)
39. Fattah, Y.E., Dechter, R.: Diagnosing tree-decomposable circuits. In: Proceedings 14th International Joint Conf. on Artificial Intelligence, pp. 1742–1748 (1995)
40. Stumptner, M., Wotawa, F.: Diagnosing tree-structured systems. Artificial Intelligence 127(1), 1–29 (2001)
41. Sachenbacher, M., Williams, B.C.: Diagnosis as semiring-based constraint optimization. In: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI), Valencia, Spain, pp. 873–877 (2004)
42. Weld, D.S.: Recent advances in ai planning. AI Magazine 20(2), 93–123 (1999)
43. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice-Hall (2010)
44. Sun, Y., Weld, D.S.: Beyond simple observation: Planning to diagnose. In: Third International Workshop on Principles of Diagnosis, Rosario, (WA) (October 1992)
45. Friedrich, G., Gottlob, G., Nejdl, W.: Towards a theory of the repair process. In: Proceedings of the Portuguese Conference on Artificial Intelligence. LNCS (LNAI), Springer, Albufeira (1991); Also appeared at the Model-Based Reasoning Workshop (AAAI 1991), Anaheim (July 1991)
46. Friedrich, G., Gottlob, G., Nejdl, W.: Formalizing the repair process. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 709–713. John Wiley & Sons, Vienna (1992); Also appeared in the Proceedings of the Second International Workshop on Principles of Diagnosis, Milano (1991)
47. Friedrich, G., Nejdl, W.: Choosing observations and actions in model-based diagnosis-repair systems. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA (October1992)
48. Wotawa, F., Nica, M., Aichernig, B.K.: Generating distinguishing tests using the minion constraint solver. In: CSTVA 2010: Proceedings of the 2nd Workshop on Constraints for Testing, Verification and Analysis. IEEE (2010)
49. Steinbauer, G., Mörth, M., Wotawa, F.: Real-time diagnosis and repair of faults of robot control software. In: Proceedings RoboCup International Symposium (2005)
50. Friedrich, G., Stumptner, M., Wotawa, F.: Model-based diagnosis of hardware designs. Artificial Intelligence 111(2), 3–39 (1999)

51. Wotawa, F.: On the Relationship between Model-Based Debugging and Program Slicing. Artificial Intelligence 135(1-2), 124–143 (2002)
52. Weiser, M.: Programmers use slices when debugging. Communications of the ACM 25(7), 446–452 (1982)
53. Mayer, W.: Static and hybrid analysis in model-based debugging. PhD Thesis, School of Computer and Information Science, University of South Australia (2007)
54. Nica, M., Nica, S., Wotawa, F.: On the use of mutations and testing for debugging. In: Software: practice & experience (in Press, 2012), `http://dx.doi.org/10.1002/spe.1142`
55. Wotawa, F., Nica, M., Moraru, I.D.: Automated debugging based on a constraint model of the program and a test case. The Journal of Logic and Algebraic Programming (in Press, 2012)
56. Kleiner, A., Steinbauer, G., Wotawa, F.: Towards Automated Online Diagnosis of Robot Navigation Software. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 159–170. Springer, Heidelberg (2008)
57. Brandstötter, M., Hofbaur, M.W., Steinbauer, G., Wotawa, F.: Model-based fault diagnosis and reconfiguration of robot drives. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and System, pp. 1203–1209 (2007)
58. Murphy, R.R.: Introduction to AI Robotics, 2nd edn. MIT Press, Cambridge (2002)
59. Hofbaur, M.W., Williams, B.C.: Mode Estimation of Probabilistic Hybrid Systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 253–266. Springer, Heidelberg (2002)