Adaptive Autonomous Systems – From the System's Architecture to Testing

Franz Wotawa

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

Abstract. Autonomous systems have to deal with situations where external events or internal faults lead to states, which have not been considered during development. As a consequence such systems have to have knowledge about themselves, which has to be used to reason about adaptions in order to fulfill a given goal. In this paper we present a control architecture that combines a the sense-plan-act paradigm and model-based reasoning. The latter is used to identify internal faults. Beside discussing the control architecture we also briefly explain advantages of model-based diagnosis where there is a shift from providing control programs to developing models. Consequently, testing of models becomes an issue. Therefore, we also introduce basic definitions related to testing models for diagnosis and discuss the involved challenges.

Keywords: Adaptive systems, model-based reasoning, testing model-based systems.

1 Introduction

Truly autonomous systems like mobile robot have to fulfill given tasks even in case of internal faults or external events potentially preventing the robot from reaching its goal. Such an autonomous robot has to detect a misbehavior by its own. Therefore, the robot has to have capabilities to reason about its state and all the consequences of the state. One consequence might be that a sensor signal cannot longer be trusted or that certain reconfigurations of the hardware and software have to be performed. The ability of reasoning about itself is even more required in cases where a robot has to deal with non-foreseen situations during the development of the robot.

Today almost no available robot is able to deal with internal faults or not previously known external events in an intelligent way. The reasons are (1) the increasing complexity of systems dealing with faults in an automatic fashion, (2) the complexity of the environment, i.e., the real physical world, of an autonomous and mobile robot, and (3) difficulties to enumerate all potential faults and interaction scenarios in advance, e.g., during the development phase of the

R. Hähnle et al. (Eds.): ISoLA 2011 Workshops, CCIS 336, pp. 76–90, 2012.
© Springer-Verlag Berlin Heidelberg 2012

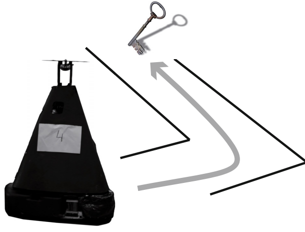

Fig. 1. Fetch the key scenario

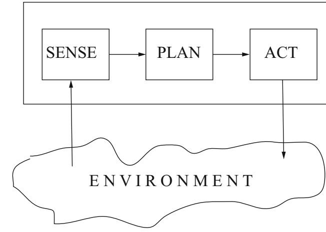

Fig. 2. Sense-plan-act control architecture

robot. The latter prevents using simple solutions like rule-based systems. Moreover, there is also the problem of testing intelligent systems especially when using neural networks and other potentially non-deterministic approaches.

In this paper we present a software architecture that allows for constructing truly autonomous systems. We use model-based reasoning techniques for enabling a robot reasoning about itself. We state the theory behind the approach. But the paper focuses more on the architecture and its components, and on testing. In the following we illustrate the underlying idea using an example scenario where a robot has to fetch a key lying on a floor (see Fig. 1). In many cases the used underlying control architecture of robots follows more or less a traditional sense-plan-act architecture depicted in Fig. 2. In this architecture the robot computes a plan using the given sensor inputs, its internal state, and the given goal. A plan in this terminology is nothing else than a sequence of actions that are executed sequentially. The execution of the actions potentially changes the state of the environment, which is sensed again. In case of failing actions or situations where a plan cannot longer be followed re-planning is performed.

A plan for our key fetching task would include actions for moving forward, turning left, and moving forward again until the robot reaches the key, i.e., $\langle forward, left, forward \rangle$. In a perfect world such a plan can be correctly executed. Let us assume now that in the first $forward$ action there is a problem with the robot's drive. The attached sensor of a wheel is returning that the wheel is not rotating. Hence, the action cannot be successfully continued and the robot might not be able to fulfill its objective anymore because of lack of functioning actuators allowing a robot to move from one position to another. However, the reason for the sensor signal might not be that the motor is not working anymore. It is also possible that the sensor is faulty but distinguishing this two explanations requires self-reasoning capabilities and a certain degree of redundancy. In this example, a robot that is able to identify wrong sensor information would be able to reach the goal and fetch the key, because the robot would not terminate plan execution. The $forward$ action would still be executed, and the robot would be able to move to the key.

In the following sections we introduce the theory behind root cause identification and discuss an algorithm. Afterwards we discuss the new robot control architecture where root cause identification is an integral part. This extended control architecture relies on combining root cause identification and the sense-

plan-act architecture. Then we discuss the software engineering challenges of such systems with respect to development and testing. Finally, we review related research and conclude the paper.

2 Model-Based Reasoning

In this section we introduce the basic concepts of model-based diagnosis (MBD) [3,4] following the definitions of Reiter [18]. The basic idea behind MBD is to use a model of the system directly for identifying the root causes of detected deviations between the observed and the expected behavior. The model specifies the correct behavior of the system. Hence, in the original definition of MBD there is no need for modeling incorrect behavior. This is especially important in cases where the incorrect behavior of components is not known in advance or too expensive to obtain. Since we also want to deal with unknown situations this capability of MBD serves our purpose.

MBD starts with a model, which is called a diagnosis system comprising a system description SD and a set of components $COMP$, where each component may fail. There are two important aspects regarding MBD to mention. First, SD comprises a set of logical sentences that state the model of the components and the structure of the system. In SD also the connections between the components, i.e., the means for communication between components, have to be specified. Second, the component models in SD define the correct behavior of the component. In order to distinguish between the correct and the incorrect behavior MBD makes use of a special predicate AB. $AB(c)$ is true if component c is faulty (or behaves abnormal). The negated predicate $\neg AB(c)$ states that c is working as expected[1].

We illustrate how to write a model for MBD using the fetch key example (Fig. 1), where we model the robot drive partially and in an abstract way. The considered part of the robot drive (see Fig. 3) comprises a motor m connected with a wheel w. The rotation of the wheel is measured using a wheel encoder e. In order to allow for deciding whether the motor is working or not we assume that the current flowing through the motor is also observed by means of a current meter c.

In the following we introduce the logical model SD written in first order logic for our example:

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. Hence, for all components X that are motors, the following logical sentence formalize their behavior:

[1] It is worth noting that it would also be possible to use a predicate NO stating normal operation instead of the negation of AB. However, the AB predicate has been used in almost all MBD papers. Therefore, we also make use of AB in order to introduce the basic MBD theory.

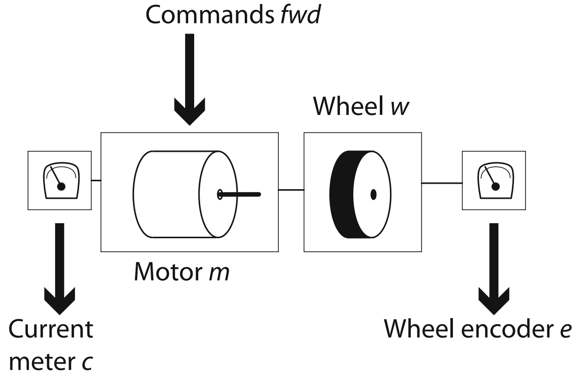

Fig. 3. Schematic of a robot drive

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

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:

$$\forall X : wheel(X) \rightarrow (\neg AB(X) \rightarrow (torque(X, fwd) \rightarrow (rotate(X, fwd) \wedge \neg resistance(X)))) \quad (2)$$

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:

$$\forall X : current_meter(X) \rightarrow \left(\begin{array}{c} \neg AB(X) \rightarrow (current(X, high) \leftrightarrow observed_current(X, high)) \\ \neg AB(X) \rightarrow (current(X, nominal) \leftrightarrow observed_current(X, nominal)) \end{array} \right) \quad (3)$$

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., $COMP = \{m, c, w, e\}$.

Having now the model $(SD, COMP)$ we are interested in finding root causes. For this purpose we have to state a diagnosis problem.

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

Before discussing solutions to diagnosis problems we should discuss some cases. In the first case assume that the model SD allows for deriving the given observations OBS, or in other words assume that the observations are not in contradiction with the model. In this case, it is reasonable to assume that the system is working as expected. The same holds for the second case where no observations are available, i.e., $OBS = \emptyset$. The third case is the more interesting one, where the observations are in contradiction with SD. In this case, we are interested in finding the root cause for the contradiction. What we have are the components and the assumptions about their correctness or incorrectness. For this purpose we use the negated AB predicate. Hence, a root cause or diagnosis should be assumptions about the logical value of the AB predicate, which lead to a logical sentence that is not in contradiction with the given observations OBS. Formally, we define a diagnosis as follows:

Definition 2 (Diagnosis). *Let* $(SD, COMP, OBS)$ *be a diagnosis problem. A set* $\Delta \subseteq COMP$ *is a diagnosis if and only if* $SD \cup OBS \cup \{\neg AB(X) | X \in COMP \setminus \Delta\} \cup \{AB(X) | X \in \Delta\}$ *is consistent.*

We illustrate this definition using our running example again. Let us assume now that a forward command is sent to the motor, a nominal current is measured, but the wheel encoder is not delivering any frequency output. The diagnosis problem includes SD and $COMP$ described previously and $OBS_1 =$

$\{cmd_fwd, observed_current(m, nominal), \neg frequency(e)\}$. Valid diagnoses are $\{e\}$, $\{w\}$, $\{c, m\}$ but also $\{c, m, e, w\}$. In order to eliminate the latter diagnosis, which states that all components are not working as expected, from the list of diagnoses we introduce the definition of minimal diagnosis.

Definition 3 (Minimal diagnosis). *A diagnosis Δ is a minimal diagnosis if there exists no diagnosis $\Delta' \subset \Delta$.*

From here on we assume that a diagnosis algorithm always returns minimal diagnoses. Hence, the 3 minimal diagnoses $\{e\}$, $\{w\}$, $\{c, m\}$ remain for our running example. This result allows for distinguishing some reasons for a misbehavior. For example, the wheel might be the root cause in case of OBS_1 but not the motor alone. In this case we might prefer small diagnosis and thus reject the fault hypothesis $\{c, m\}$. However, we are not able to distinguish the diagnosis $\{w\}$ from $\{e\}$. In order to solve this challenge there are 3 options: (1) introduce new observations, which might be not possible in case of autonomous systems with a limited number of sensors, (2) use fault probabilities, and (3) extend the model to eliminate diagnoses.

Option (2) can be implemented by assigning a fault probability $p_f(X)$ to each component X. In this case we are able to compute the probability of a diagnosis assuming independence of component faults as follows:

$$p(\Delta) = \prod_{X \in \Delta} p_f(X) \cdot \prod_{X \in COMP \setminus \Delta} (1 - p_f(X)) \tag{7}$$

If we assume that a fault in a wheel is less likely than a fault in the wheel encoder, Equation 7 returns a higher probability value for the diagnosis *wheel encoder* than for the diagnosis *wheel.* It is worth noting that fault probabilities might also change over time. For a theory of adapting fault probabilities using diagnosis information we refer the interested reader to [6].

In option (3) the idea is to eliminate diagnoses by adding knowledge about impossibilities to the system description. Friedrich et al. [10] introduced this concept of *physical impossibilities* and explained the advantage with respect to diagnosis capabilities. For our running example, we might state that it is impossible that the wheel is the reason for observing no frequency in case of nominal current flow through the motor. In other word we state that the physical connection between the motor and the wheel can never break. This is not true in general but might be applicable in special cases. In order to state this impossibility we defined SD' to be SD together with the rule $current(m, nominal) \wedge \neg rotate(w, fwd) \rightarrow \bot$. When using SD' together with OBS_1 only diagnosis $\{e\}$ remains as single diagnosis.

There are many algorithms available for computing diagnosis. Reiter [18] introduces a hitting set based algorithm that makes use of conflicts, which was corrected by Greiner et al. [12]. De Kleer and Williams [6] introduced their general diagnosis engine (GDE). GDE is based on an assumption based truth maintenance system (ATMS) [5]. Later on Nejdl and Fröhlich [11] presented a very fast diagnosis engine that is also based on logic model representations.

Diagnosis algorithms that make use of certain structural properties include [8] and [22].

Note that there is no restriction regarding the modeling language used for model-based reasoning. In this section we make use of first order logic, but also constraint representations [7], or other formalisms might be used. The only requirement is to have a solver that allows for checking consistency. Algorithm 1 introduces a straight forward but not optimal algorithm that computes minimal diagnoses up to a pre-defined size. The algorithm basically checks all subsets of $COMP$ for being a diagnosis. Although there are many improvements possible the algorithm is fast enough if we are interested in smaller diagnoses like single faults only.

Algorithm 1. DIAGNOSIS

Input: A model $(SD, COMP)$, a set of observations OBS, and the maximum size of computed diagnoses $max > 0$. In case max is not given, we assume $max := |COMP|$.
Output: A set of minimal diagnoses accordingly to Definition 3.

{Computes all minimal diagnoses up to the given size.}
Let DS be the empty set.
for i:= 0 **to** max **do**
  Generate all subset of $COMP$ of size i and store the results in DS'.
  Remove all elements in DS' that are subset of minimal diagnosis stored in DS.
  for all $\Delta \in DS'$ **do**
    Call the theorem prover on $SD \cup OBS \cup \{\neg AB(X) | X \in COMP \setminus \Delta\} \cup \{AB(X) | X \in \Delta\})$.
    if the theorem prover call returns an inconsistency **then**
      Remove Δ from DS'.
    end if
  end for
  Let DS be $DS \cup DS'$.
end for
return DS

3 A Model-Based Control Architecture

In this section, we discuss the extension of the ordinary sense-plan-act control architecture (Fig. 2) to handle cases of internal faults and external events, preventing actions to be executed. In the original architecture a plan is generated that leads from an initial state to a goal state. The state (comprising the sensor information and internal state) is observed and used to select actions based on the plan. The actions are executed using the actuators. This execution might lead to changes in the internal state or in the environment. As discussed in the introduction the original control architecture cannot react on internal faults and hardly interact on external events, preventing actions to be executed and therefore plans to reach their goals.

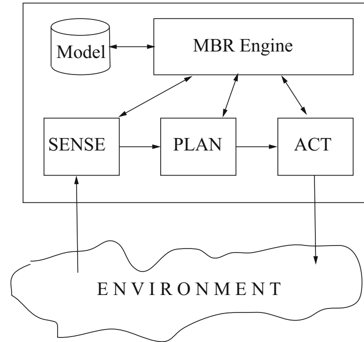

Fig. 4. The adaptive robot control architecture

In order to overcome this problem, we introduce an extended sense-plan-act control architecture, which computes root causes in case of a detected misbehavior. In Figure 4 the extended sense-plan-act control architecture is depicted.

The main control loop of the sense-plan-act architecture remains the same but in some cases, which we discuss later, a model-based reasoner is used to decide the current health status of the autonomous system. The reasoner has access to the internal state of the controller and the sensor information. The health status needs not to be always checked. A check has to be performed only in cases where an action that is decided by the planning module cannot be successfully executed. As already stated, there might be several reasons: (1) there is a fault in the hardware preventing the action to be successfully terminated, or (2) an external event occurred. The latter requires re-planning activities whereas the first reason requires determining the fault in order to adapt the behavior.

Algorithm 2 describes the underlying behavior of the extended sense-plan-act architecture. We assume that the goal state specifies the purpose of an autonomous system. The task of the control architecture is now to find a plan, i.e., a sequence of actions, that allows the system to reach the goal state from the current state, and to execute this plan. This idea originates from STRIPS planning [9] and our algorithm relies on the same input data, i.e., a planning knowledge base M_p comprising a set of actions a where each action has a corresponding set of pre conditions and effects. Moreover, we require a goal state S_G and a diagnosis model $(SD, COMP)$, used in cases of failing actions during plan execution.

In Algorithm 2 we make heavily use of a function SENSE(). This function returns the current state of the autonomous system comprising the internal state and the information obtained from the sensors. We assume that the sensor information is only given when reliable. Hence, in case of a diagnosis indicating a sensor to fail, the information is no longer provided. Although time is not handled explicitly in the algorithm it is worth noting that time exceeds during execution. Therefore, SENSE() represents the state at a discrete point in time only, where measurements and the internal state are observed. We assume that this is done on a regular basis.

Algorithm 2. EXT_PSA_ARCH

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

```
Let p := PLAN(M_p,SENSE(),S_G).
while p is not empty do
  Let a be the first action of plan p.
  Remove a from p.
  if pre conditions of a are not fulfilled in SENSE() then
    Let p := PLAN(M_p,SENSE(),S_G).
  else
    Execute a.
    if execution terminates with a failure then
      Let S_Δ be the result of calling DIAGNOSIS(SD,COMP,SENSE()), and Δ be
      the leading diagnosis of S_Δ.
      if Δ indicates a sensor failure only then
        Consider a to be executed without failure and proceed.
      else
        Remove all actions from the planning model that cannot be longer used
        because of diagnosis Δ.
        Let p := PLAN(M_p,SENSE(),S_G).
      end if
    else
      if effects of a are not fulfilled in SENSE() then
        Let p := PLAN(M_p,SENSE(),S_G).
      end if
    end if
  end if
end while
```

The control algorithm starts computing a plan based on the available information, i.e., the current state provided by SENSE() 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 might happens 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 from which a leading diagnosis can be obtained. A leading diagnosis in our case might be 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, measures for distinguishing diagnoses have to be performed, like providing testing procedures. For example, Wotawa et al. [25] 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.

The extended sense-plan-act control architecture provides adaptation due to internal faults and external events. Depending on the used models it allows for distinguishing sensor and actuator faults. It might also be used for handling faults in software providing a model of the behavior. Of course the implementation of the control loop and the diagnosis module have to be correct and cannot be corrected using the proposed approach.

4 Testing Model-Based Systems

One advantage of the model-based reasoning approach is that the underlying implementation, i.e., the diagnosis module and the control architecture, can be easily re-used. The only changes necessary are changes of the underlying models. Therefore, we focus on challenges of testing models instead of testing the implementation of the different modules that together contribute to the overall system's architecture. Testing models especially when they are written in logic or any other relational language is rather similar to ordinary software testing [1]. In both cases we are interested in comparing the behavior imposed by a system with the expected behavior. However, there are some differences and challenges like answering the question when to stop testing? Usually, in software engineering practice there are some criteria like mutations scores or coverage, e.g., statement or branch coverage, a test suite has to fulfill. In case of testing models such criteria can hardly be applied directly and new criteria have to be developed.

We start with discussing testing the diagnosis model. What we have to ensure is that the model allows for deriving all correct diagnoses for any given set of observations. Moreover, the model should never allow deriving diagnoses that are not correct. So what we want to show is whether DIAGNOSIS(SD,$COMP$,OBS) $= DS_{expected}$ holds for all possible OBS or not. This is of course not feasible even in the case of a finite number of predicates that might be used in OBS. Therefore, in testing diagnosis models we have to restrict the number of tests. Formally, we introduce a test suite for a model $(SD, COMP)$ as follows:

Definition 4 (Test, test suite). *Let* $(SD, COMP)$ *be a diagnosis model and* PO *the set of possible observations. A tuple* (OBS, DS, max)*, where* OBS *is*

a set of observations, DS a set of diagnoses with cardinality less or equal to max, is a test for the diagnosis model if $OBS \subseteq PO$ *and* $SD \cup OBS \not\models \bot$ *or if* $OBS \subseteq PO$, $SD \cup OBS \models \bot$ *and* $DS = \emptyset$ *. A test suite for the diagnosis model is a set of tests.*

Note that in the definition $\models$ stands for the logical consequence, and $\bot$ for the contradiction. In the above definition there is no restriction on DS with the exception of the case, where the observations directly are in contradiction with the model. In this case no diagnosis is allowed to be computed. This fact follows directly from the definition of diagnosis in MBD.

We are now able to specify soundness and completeness of a model with respect to the given test suite. In particular, we might be interested whether a given model delivers all correct diagnoses. This can be ensured when having sound and complete models. A model is sound if only correct diagnoses can be obtained from the model. A model is complete if all diagnoses can be derived from the model and the given observations.

Definition 5 (Soundness, completeness). *Let* $(SD, COMP)$ *be a diagnosis model and* TS *its non empty test suite.* $(SD, COMP)$ *is sound with respect to* TS *iff for all* $(OBS, DS, max) \in TS$*: DIAGNOSIS(SD,COMP,OBS,max)* $\subseteq DS$*.* $(SD, COMP)$ *is complete with respect to* TS *iff for all* $(OBS, DS) \in TS$*: DIAGNOSIS(SD,COMP,OBS,max)* $\supseteq DS$*.*

We further say that a diagnosis model is correct with respect to a test suite iff it is sound and complete.

After defining test suites and their impact on the model under test, we are interested in generating test suites that obey a reasonable criterion. Although we know that *"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."*[2] generating tests that at least allows us to check correctness for some important cases is essential. Hence, we have to ask ourselves what are the important cases for diagnosis? When considering the practical application of diagnosis, correctly identifying single fault might be enough. Double and triple faults might occur but can hardly be distinguished. Therefore, there would be a strong need for handling a huge amount of diagnoses concurrently, which increases complexity. Hence, classifying test suites regarding their capabilities of testing diagnosis results up to a certain cardinality, seems to be a good choice.

We now generalize the concept of classifying test suites with respect to the checking diagnoses up to a certain cardinality.

Definition 6 (n-cardinality criteria). *A test suite* TS *for a diagnosis model* $(SD, COMP)$ *fulfills the* $n-$*cardinality criteria iff* $\forall \Delta \subseteq 2^{COMP} \wedge |\Delta| \leq n : \exists (OBS, DS, max) \in TS : (\Delta \in DS \vee \exists \Delta' \in DS : \Delta \supset \Delta')$*.*

Obviously in this definition a test suite that fulfills 2-cardinality also fulfills 1-cardinality. Therefore, when constructing a test suite, starting with 1-cardinality

[2] from Edsger Wybe Dijkstra, The Humble Programmer, ACM Turing Lecture 1972.

and than extending the test suite to fulfill the higher cardinality criteria, seems to be a good procedure. It is worth noting that even constructing 1-cardinality test suites may not be easy because specifying all diagnoses of size 1 for given observations can be a hard problem. Hence, in practice it might be a good option to iteratively construct test cases. First, start with observations that must lead to some diagnosis. Call the DIAGNOSIS algorithm for computing all diagnosis up to a given size. If the intended diagnoses are part of the result, check the remaining diagnoses (if there are any). Remove all diagnoses that are unexpected. Add diagnoses that are missing and generate the new test case.

Besides testing the diagnosis model the planning model and the interaction of the planning module and the diagnosis module has to be tested. The planning model might be tested similarly to the diagnosis model. The initial and the goal state have to be specified, and possible plans have to be provided. If the planner computes the same plans, the model has to be correct with respect to the test case. Again certain corner cases should be specified and tested in order to ensure at least partial correctness.

The interaction between the planner and the diagnosis engine can be tested by stating more complex scenarios. The integration test has to specify a scenario where a fault occurs, which has to be detected by the diagnosis algorithm. If the implementation together with the model fulfill the scenarios the model might be assumed to be correct because the system behaves like expected. However, because of complexity it is impossible to check all scenarios. Again only some important cases can be used for evaluation purposes.

Testing model-based applications basically means testing the developed models. Because of the complexity of models and underlying domains it is a painstaking task to generate larger sets of test cases. Moreover, the quality of test suites can hardly be quantified. In case of the model-based diagnosis module we are able to quantify test suites with respect to the cardinality of diagnoses handled in a certain case. What remains to do is to work on supporting test case generation for model-based systems. Even for diagnosis this seems not to be an easy task and requires an iterative procedure.

5 Related Research

The application of MBD in the context of autonomous mobile systems is not new. To our knowledge Williams and colleagues [24,17] were the first introducing a system that automatically adapts to internal faults in order to reach its objectives. The MBD system had been tested in real-world conditions as part of the control system of the NASA deep space one space probe[3]. The focus of the approach was on hardware faults. Hence, intelligent interactions in case of external events were not considered. Moreover, the space probe itself had no intended capabilities of changing the external world. This is different to the mobile robotics domain where actions have an impact on the environment.

[3] See `http://nmp.nasa.gov/ds1/`

Approaches for applying MBD to the domain of robotics include [2], [15], and [16] where the focus lies more in diagnosing interactions between many mobile robots that should work together in order to reach a specified goal. Steinbauer and colleagues [19] introduced a system that is capable of dealing with software faults in mobile robot control systems. The approach is based on a very simplified structural model of the control software and makes use of monitoring procedures for detecting a misbehavior. The structural model is used to extract pieces of software, i.e., modules, that should be restarted in order to overcome the detected problems. The approach of Steinbauer and colleagues does not make use of a behavior model and does not consider the case of external events.

In [14] the authors describe a MBD approach for handling faults in robot drives. In particular the authors claim that they can adapt the kinematics of the robot in case of faults in the drive. In the paper the focus is on how to adapt the kinematics. The whole system's architecture is explained but not in detail. The adaption of high level control is also not given in detail. However, we borrow the idea of using diagnosis information for changing high-level control from [14].

The idea of adapting plans based on diagnosis knowledge gained during execution was outlined in [23] and [21]. The first paper deals in particular with dependent failures, which might occur in robotics systems. In both papers the authors make use of repair in case of misbehavior to ensure robustness. In this paper we formally introduce the system's architecture and also discuss the challenge of verifying such systems by means of model testing.

The work most closely to our work described in this paper is [13]. There the authors introduce a MBD approach for mobile autonomous systems that is capable to handle internal faults as well as external events. The approach of Gspandl et al. makes use of a Reiter's situation calculus and is characterized by combining control and diagnosis knowledge in a uniform framework. In contrast to this work our approach separates planning and control from diagnosis. Moreover, we also tackle the problem of testing such systems, which has not been covered before.

Steinbauer and Wotawa [20] discussed the evaluation of adaptive mobile and autonomous robots. There the objective is to find an evaluation testbed that allows for comparing different solutions. The paper does not deal with testing such systems during development. Instead the paper focuses on the validation of intelligent and adaptive robots with respect to their robustness in practical situations.

6 Conclusion

In this paper we introduced a control architecture that enables adaptive control of autonomous systems. The control architecture integrates the sense-plan-act architecture and a model-based diagnosis engine. As a consequence both external events and internal faults can be treated in a smart way. Internal faults are handled by the diagnosis engine directly. External events cause re-planning activities. An advantage of the proposed system is also the implicit software re-use.

The implementation of the approach could be used in many applications. Only the underlying models have to be adapted or changed.

From the software engineering point of view, there is a focus switch from coding to modeling. Or in other words, model-based development requires writing models instead of more traditional source code. Consequently, testing has to focus on model testing. Unfortunately, testing models is not an easy task. In the paper we outline some basic definitions of testing models for diagnosis and briefly discuss an iterative approach for generating test suites from models.

Although, authors have shown in their papers that the model-based reasoning approach really allows for generating adaptive autonomous systems, there is almost no work on developing and testing such systems. In future work we will discuss this issue in much more detail. It is also worth mentioning that the proposed approach requires a certain degree of redundancy. This redundancy is different from the redundancy used to ensure robustness and reliability of systems, where components are typically available many times. In case of MBD there is a need for an overlap of functionality, e.g., sensor input, in the system that is maybe provided by different sensors. Developing guidelines for constructing systems in order to fulfill the requirement of partial redundancy is also an open research issue.

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

References

1. Beizer, B.: Software Testing Techniques. Van Nostrand Reinhold (1990)
2. Daigle, M., Koutsoukos, X., Biswas, G.: Distributed diagnosis of coupled mobile robots. In: IEEE International Conference on Robotics and Automation, pp. 3787–3794 (2006)
3. Davis, R.: Diagnostic reasoning based on structure and behavior. Artificial Intelligence 24, 347–410 (1984)
4. Davis, R., Hamscher, W.: Model-based reasoning: Troubleshooting. In: Shrobe, H.E. (ed.) Exploring Artificial Intelligence, ch.8, pp. 297–346. Morgan Kaufmann (1988)
5. de Kleer, J.: An assumption-based TMS. Artificial Intelligence 28, 127–162 (1986)
6. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artificial Intelligence 32(1), 97–130 (1987)
7. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
8. El Fattah, Y., Dechter, R.: Diagnosing tree-decomposable circuits. In: Proceedings 14th International Joint Conf. on Artificial Intelligence, pp. 1742–1748 (1995)
9. Fikes, R.E., Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, 189–208 (1971)
10. Friedrich, G., Gottlob, G., Nejdl, W.: Physical impossibility instead of fault models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Boston, pp. 331–336 (August 1990); also appears in Readings in Model-Based Diagnosis. Morgan Kaufmann (1992)

11. 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)
12. 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)
13. Gspandl, S., Pill, I.H., Reip, M., Steinbauer, G., Ferrein, A.: Belief management for high-level robot programs. In: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI (2011)
14. Hofbaur, M., Köb, J., Steinbauer, G., Wotawa, F.: Improving robustness of mobile robots using model-based reasoning. Journal of Intelligent & Robotic Systems 48(1), 37–54 (2007)
15. Kalech, M., Kaminka, G.A.: On the design of coordination diagnosis algorithms for teams of situated agents. Artificial Intelligence 171(8-9), 491–513 (2007)
16. Micalizio, R., Torasso, P., Torta, G.: On-line monitoring and diagnosis of a team of service robots: A model-based approach. AI Communications 19(4), 313–340 (2006)
17. Rajan, K., Bernard, D.E., Dorais, G., Gamble, E.B., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P.P., Rouquette, N.F., Smith, B.D., Taylor, W., Tung, Y.: Remote agent: An autonomous control system for the new millennium. In: 14th European Conference on Artificial Intelligence (ECAI), pp. 726–730 (2000)
18. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
19. Steinbauer, G., Mörth, M., Wotawa, F.: Real-time diagnosis and repair of faults of robot control software. In: RoboCup International Symposium, pp. 13–23 (2005)
20. Steinbauer, G., Wotawa, F.: Evaluating the robustness of the perception-decision-execution cycle of autonomous robots. In: Proceedings of the ICAR Workshop on Performance Measures for Quantifying Safe and Reliable Operation of Professional Service Robots in Unstructured, Dynamic Environments (2011)
21. Steinbauer, G., Wotawa, F.: Robust plan execution using model-based reasoning. Advanced Robotics 23(10), 1315–1326 (2009)
22. Stumptner, M., Wotawa, F.: Diagnosing Tree-Structured Systems. In: Proceedings 15th International Joint Conf. on Artificial Intelligence, Nagoya, Japan (1997)
23. Weber, J., Wotawa, F.: Diagnosis and repair of dependent failures in the control system of a mobile autonomous robot. Applied intelligence (2008)
24. Williams, B.C., Pandurang Nayak, P.: Immobile robots – ai in the new millennium. AI Magazine, 16–35 (1996)
25. 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)