A Hierarchical Monitoring and Diagnosis System for Autonomous Robots

Gerald Steinbauer-Wagner, Leo Fürbaß, Marco De Bortoli

Institute for Software Technology
Graz University of Technology, Graz, Austria
Email: steinbauer@ist.tugraz.at

Abstract

This paper addresses the need for autonomous robots to achieve flexible goals in dynamic environments. We propose a hierarchical diagnosis concept for layered control architectures, ensuring consistent information and reliable decision-making. Our approach addresses challenges like failing actions, uncertain observations, and unmodeled events by propagating observations and diagnoses throughout the hierarchy. This enhances adaptability and dependability in various domains.

I. Introduction

There is an increasing demand for autonomous robots that can pursue flexible goals in an open and dynamic context. This asks for true autonomy, requiring minimal or no human intervention. In contrast, even commercial robot systems often fail to pursue their goals and interact with an unpredictable dynamic environment. There are various reasons for this shortcoming such as failing actions, not modeled or observed changes in the environment, or issues in the perception. In this paper, we propose a hierarchical diagnosis concept for a control architecture for autonomous robots. Such architectures are usually organized in layers. When progressing from one layer up to the next usually information is abstracted and the scope of the decision-making is narrowed conceptually and widened temporally. Moreover, the higher layer uses the lower one to refine and execute its decisions. In this interplay information, commands, and execution results need to be exchanged while performing abstraction and concretization respectively. In order to allow dependable decision-making and execution the knowledge and information on the various levels needs to be kept consistent with the evolution of the environment. A standard architecture is the three-tier architecture that combines a deliberative layer with planning capabilities, an executive layer able to execute intermediate tasks by refining them into skills (atomic actions), and a skill layer able to handle the execution of such skills [12]. Monitoring and diagnosis need to be performed on the various levels in the corresponding layer context. The main challenges for an intelligent agent and its control architecture performing non-trivial tasks in dynamic environments are: (1) skills (actions) that are not able to establish all its intended effects, (2) observations about the environment that are uncertain or wrong, (3) exogenous events changing the environment in a way the agent has not modeled or is not aware of, and (4) failed or interrupted skills (actions) where the resulting situation is not entirely clear. All these challenges may render the local context of the various levels inconsistent jeopardizing dependable decision-making and execution. In order to address these challenges we propose a holistic hierarchical diagnosis concept that propagates local observations, execution results, and diagnoses up and down the control hierarchy to allow reasoning with a broadening context if issues cannot be detected or handled in the corresponding layer. The diagnosis concept presented in this paper augments our planning and execution framework, which was designed in a general way and can be easily adapted to different domains.

II. The control architecture

In this section, we briefly describe the general architecture of the control framework. It resembles the general structure used for multi-robot systems, and the main parts of the software are divided into a three-level hierarchy, as shown in Figure 1. The most abstract layer, namely the High-Level, involves task generation, assignment, and coordination between robots. It is responsible for the long-term strategy and monitoring of it. The Mid-Level is concerned with task refinement and execution by a specific agent through Behavior Trees [7]. Given a High-Level action, this layer transforms it into the corresponding Behavior Tree. The Behavior Tree decomposes the actions into a set of subtasks, represented as an extended state machine. To give an insight, consider a high-level action of passing through a closed door. Such action will be transformed into a set of different subtasks by the Mid-Level. Passing to the door might be represented by a sequence of checking the door state, opening the door, passing it, and finally closing it. This level of detail is neither useful nor helpful for the High-Level. The Low-Level consists of the robot platform, i.e., sensing, low-level monitoring and control of actuators as well as basic skills such

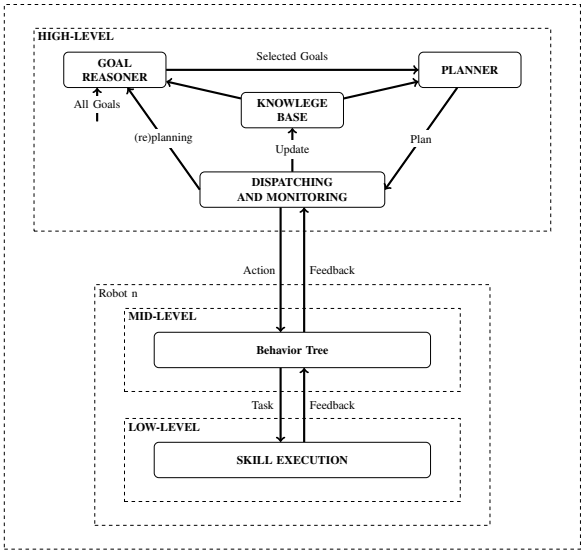

Fig. 1. Basic control architecture.

as localization and navigation. Here the subtasks generated at the Mid-Level are further refined.

A. The High-Level Planning and Dispatching System

The Planning and Dispatching Framework implements a control strategy for multi-agent systems in dynamic domains. It is based on three main components: (1) a Goal Reasoner, (2) a Planner, and (3) a Dispatching and Monitoring system. In Figure 2, the interaction between the components is shown. The Dispatching and Monitoring component plays the role of the main controller that invokes the Goal Reasoner [1], and consequently the planner, and executes the obtained plan, by sending the tasks to the Mid-Level which builds the corresponding Behavior Trees. The plan execution is constantly monitored for issues that may require regeneration of the goals of the plan, e.g., failed actions, deadline violations, or external events. The planner module is based on Temporal PDDL [11]. Given a fixed domain, namely the description of possible actions, and the instance, generated from the Knowledge Base, the result of the planning process is a temporal plan, which is represented by the schedule σ, formed by a set of triples $\langle a, t_a, d_a \rangle$, where a is an action, t_a is the time when the action a needs to be started, and d_a is the duration of the action. The Knowledge Base, namely the set of beliefs about the status of the world, is a crucial component. For easier integration with PDDL planning, the Knowledge Base reflects PDDL syntax. It consists of a set of atoms. An atom is a grounded predicate. For instance, the atom $(at\ r\ l)$ specifies that the robot r is at location l. Each Temporal PDDL action of the plan features a set of preconditions and effects. The preconditions of an action a are formed by three sets of atoms: the start condition $cond_{\vdash}(a)$, the invariant condition $cond_{\leftrightarrow}(a)$, and the end condition $cond_{\dashv}(a)$. The effects are instead divided into the start positive (add) effects $eff^{+}_{\vdash}(a)$, the start negative (remove) effects $eff^{-}_{\vdash}(a)$, the end positive (add) effects $eff^{+}_{\dashv}(a)$ and the end negative (remove) effects $eff^{-}_{\dashv}(a)$. Every time an action is dispatched, the Knowledge Base KB is checked for consistency by making sure that $cond_{\vdash}(a) \in KB$ and $cond_{\leftrightarrow}(a) \in KB$. If it holds, we then apply the starting effect of the action, with $KB \leftarrow KB \cup eff^{+}_{\vdash}(a)$ and $KB \leftarrow KB \setminus eff^{-}_{\vdash}(a)$. The same procedure is applied for end actions, considering end conditions and effects instead. ROSPlan follows a similar approach [5]. As a consequence, in case of failures or mismatches between the Knowledge Base and the real world, it is crucial to diagnose the problem accurately and derive the corresponding effects on the set of beliefs. In fact, an inconsistent Knowledge Base may compromise the entire decision-making. In fact, by using diagnosis and monitoring, it may be possible to track the specific problem in the interaction of the agent with its environment and fix the KB accordingly, allowing it to pursue its goals. Therefore, the diagnosis system presented in the next chapter has been developed.

B. Control and Recovery Strategy in the Mid-Level

The use of Behaviour Trees (BT) in Mid-Level not only allows realizing action refinement easily but also the integration with supervison and recovery strategies. Conditions, fallbacks, and other predefined types of nodes allow for an easy implementation of both supervision and recovery. For this, the original BT that refines an action is transferred into a dependable failsafe version, where patterns for fault monitoring and mitigation are encoded using domain/expert knowledge. An example is the use of a ReactiveSequence node, to continuously check guarding conditions. Moreover, this pattern is a subtree of a BT having a RetryUntilSuccessful node as the root. This node retries to execute the whole subtree in case of failure, within a predefined maximum number of attempts. This allows the implementation of a simple local recovery strategy. If recovery fails on this level, the problem has to be handled either through local diagnosis or by the deliberative layer.

III. Hierarchical Monitoring and Diagnosis for Decision Making and Execution

The use of an architecture with various levels of abstraction, necessary to tackle complex dynamic domains requiring planning, synchronization, and skill

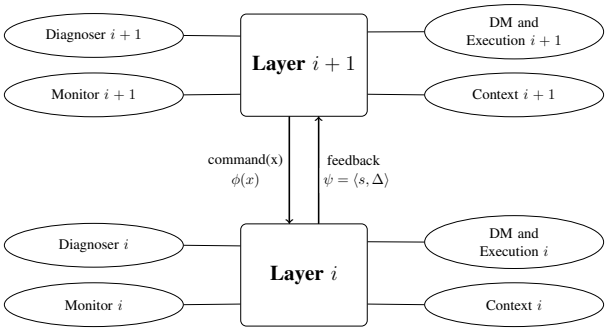

Fig. 2. Hierarchical Interaction, Monitoring, and Diagnosis.

execution, poses some challenges regarding the implementation of a holistic monitoring and diagnosis system. In order to correctly detect and identify problems in task planning and execution, the system must be aware of the different layers and knowledge representations used at the different layers. Moreover, it needs to understand how the different layers interact and whether there is already a local mechanism for fault detection and handling on some layers sufficient to deal with a problem. Hierarchical diagnosis is a well-known concept, to speed up diagnosis in complex systems using abstraction and clustering [6]. In our case, the challenge is to combine information and diagnosis results from different levels of the architecture into a globally consistent diagnosis. An interesting aspect regarding diagnosis is that the temporal context and the conceptual context show an inverse proportional nature along the levels. While the level of granularity of facts shrinks when moving up the hierarchy (abstraction of actions), the temporal context becomes wider (a sequence of actions versus only one behavior). In order to address these challenges we propose a hierarchical monitoring and diagnosis system where the interaction between a layer $i+1$ and i follows the pattern shown in Figure 2. We assume that each layer i possesses its individual set of the following components: (1) context, (2) decision-making and execution (DME), (3) monitoring, and (4) diagnosis. The implementation of the individual components depends on the layer. In the subsequent sections, we will provide details for the layers used in our architecture. The context holds all information relevant for decision-making and execution. We assume the context is represented using some language $\mathcal{L}_i$. DME can come up with a decision on what activities need to be performed to achieve a given goal or command and can execute that decision. The monitor supervises the correct execution of the decision while the diagnoser provides explanations of why an execution failed. DME might be able to use those diagnoses to reconfigure the decision and execution or propagate them to the higher layer if it is not able to handle it sufficiently. In order to represent the interaction between layer $i+1$ and i we assume that layer $i+1$ issues commands represented in $\mathcal{L}_{i+1}$. For layer i we assume a function $\phi : \mathcal{L}_{i+1} \rightarrow \mathcal{L}_i$ which performs the concretization of that command. Layer i will work on the achievement of that command. During the handling of the command intermediate observations o represented in $\mathcal{L}_i$ will be propagated to layer $l+1$ that can use these observations for monitoring. In order to allow an integration of o in the context $i+1$ we assume an abstraction function $\psi : \mathcal{L}_i \rightarrow \mathcal{L}_{i+1}$. If the processing of the command concludes (nominal or abnormal) the feedback tuple $\langle s, \Delta \rangle$ where $s \in \{success, fail\}$ represents the execution result and Δ (represented in $\mathcal{L}_i$) potential explanations for a failed execution. A reported failure occurs if a monitor at layer i detects a problem or if a lower layer's issue is propagated and unresolved. Even reported success may not mean correct completion, as broader context issues might only be detectable in higher layers. Thus, higher-layer monitors must verify lower-layer execution results.

A. Skill Layer (Low-Level)

The lowest layer is responsible for the execution of atomic skills like navigation or manipulation. As usual, such a layer is implemented using the Robot Operating System (ROS) [18]. Besides means of structuring data communication (topics) and computation (nodes), it provides concepts for implementing behaviors (skills) via the action concept. It allows running a concurrent behavior such as navigating to a goal [16] while reporting intermediate feedback on the progress as well as the final result of the skill execution. Standard implementations such as the ROS navigation stack contain hand-coded monitors (e.g. tracking the progress of the robot pose) but do not provide detailed explanations in the case of a failure. Diagnosers such as the one proposed in [9] that can provide detailed explanations about a navigation fault need to be integrated. Moreover, only rudimentary recovery behaviors are implemented in the navigation stack. Thus, a diagnosis and integration into the corresponding context need to be done at the higher layers. Intermediate observations might be provided by dedicated perception modules such as a localization module that tracks the robot pose [21]. The context and observations of the skill layer is built up by continuous values representing circumstances such as the robot pose ($\langle x, y, \theta \rangle$). Also, more detailed information such as the particle set in the localization can be used to create a monitor [10].

B. Executive Layer (Mid-Level)

The task of the executive layer is to refine the abstract actions dispatched by the deliberative layer and supervise the execution of the refined actions. While it is common to use the concept of hierarchical

task networks (HTNs) [17] or BDI architectures [14] for the implementation we use the concept of Behavior Trees [8] because of its simpler use and ROS integration. In contrast to a declarative approach such as planning BTs are composed of different nodes (e.g. conditions, sequence, fallback, atomic action) and represent a control policy. The execution leads to a trace of atomic actions that is dependent on the actual development of the context. Any legal termination of a tree counts as a success. If any node in the tree fails a fail is reported. The issue is that BTs can replicate an entire imperative programming language, necessitating runtime verification and full operational semantics for complete diagnosis [20, 3]. For now, we consider a BT as an ordered sequence of sub-tasks for the sake of simplicity. After the execution of each sub-task, feedback from the low-level is returned, stating the successful or failed execution of the task. In case of failure, the following subtasks are not executed, and the Mid-Level notifies the failed execution of the PDDL action to the High-Level. In case of a successful execution of the entire Behavior Tree, the Knowledge Base is updated at the High-Level with the effect of the PDDL action. However, determining which effects have to be applied and which do not in case of a failure in the execution of the BT is not trivial. In order to make the KB consistent, we need to know the nature of the failure and how it affects the other elements of the world. For this reason, a local diagnosis needs to be performed on the Behavior Tree, supported by observations made at the Mid-Level and the structure of the entire BT execution. Moreover, such a technique is applied even in the case of positive feedback from the Low-Level. Some failures may not have been detected at the Low-Level because of its limited context. The application of a diagnosis technique, which takes into consideration a broader set of observations, may help to detect hidden faults in the execution.

C. Deliberative Layer (High-Level)

As described above a High-Level PDDL action is refined at the Mid-Level into a Behavior Tree. The diagnoser on the high level is based on History-Based Diagnosis [15] and follows the idea presented in [13]. It is triggered either by the monitor at its level or by failure reports and diagnoses reported from the lower levels. The monitor is currently a simple consistency check between the Knowledge Base and reported observations. As pointed out above the Knowledge Base is updated with an action's effects in the case of its successful execution or by the integration of observations from the lower levels. In the latter case we follow the concept of sensing actions [19] that can override facts in the Knowledge Base. Once an inconsistency is triggered the diagnoser aims to find an alternative sequence of the actions executed so far whose effects are consistent with the Knowledge Base and the recently made inconsistent (unexpected) observation. Alternatives are generated by either exchanging actions with a variant modeling a particular fault mode of that action or by adding actions that model the occurrence of exogenous events. The downside of this approach is that it is computationally extremely demanding and explicit models of all action faults and exogenous events are needed. If a consistent execution trace is found, the Knowledge Base is updated according to the effect of its actions.

IV. What is missing?

The integrated architecture proposed in this paper is still a work in progress. Nevertheless, the proposed control architecture and the proposed monitoring and diagnosis schema had been implemented as a proof of concept and successfully used in domains such as the RoboCup Logistics League (RCLL) [4]. The limitation here is that the approach was implemented ad-hoc with mostly separate concepts on the individual levels and a full formal integration in a truly hierarchical diagnosis approach is still missing. In fact, during RCLL games, our diagnostic approach ensures robust and solid execution, preventing permanent production halts and always allowing for some recovery strategy. However, the lack of better integration of diagnostics across different levels sometimes necessitates aggressive recovery strategies, potentially discarding recoverable products unnecessarily. The difficulty is here still the proper mapping of the individual representations of contexts, decision-making approaches, execution semantics, diagnosers, and monitors a unified representation. A promising way is shown in [2] where the Situation Calculus and the agent language ConGolog as a unified representation across levels in agent supervision. Unfortunately, such methods rely heavily on reasoning and are therefore computationally very expensive. Moreover, also challenges remain on the individual levels such as the proper monitoring and diagnosis of complex Behavior Trees because of the operational semantics. Another important aspect is the reuse of domain knowledge and common sense knowledge which are of utmost importance for feasible monitoring and diagnosis. Here still a lot of manual work is needed hindering the automation of the reuse and also limiting the amount of usable knowledge. This is currently particularly interesting as we see Large Language Models (LLMs) as compiled common sense knowledge that is richer and easier to access as attempts like Cyc.

With this workshop paper, we like to stimulate a discussion about the idea of an integrated monitoring and diagnosis approach for cognitive architecture and look forward to valuable feedback on our ideas.

References

[1] David W. Aha. Goal Reasoning: Foundations, Emerging Applications, and Prospects. *AI Magazine*, 39(2):3–24, June 2018.

[2] Bita Banihashemi, Giuseppe De Giacomo, Yves Lespérance, et al. Hierarchical Agent Supervision. In *AAMAPROCEEDINGS OF THE INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS*, volume 2, pages 1432–1440. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2018.

[3] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018.

[4] David Beikircher, Marco De Bortoli, Leo Fürbaß, Thomas Kernbauer, Peter Kohout, Dominik Lampel, Anna Masiero, Stefan Moser, Martin Nagele, and Gerald Steinbauer-Wagner. Robust Integration of Planning, Execution, Recovery and Testing to Win the RoboCup Logistics League. In Cédric Buche, Alessandra Rossi, Marco Simões, and Ubbo Visser, editors, *RoboCup 2023: Robot World Cup XXVI*, pages 362–373, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-55015-7.

[5] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, 2015.

[6] Luca Chittaro and Roberto Ranon. Hierarchical model-based diagnosis based on structural abstraction. *Artificial Intelligence*, 155(1):147–182, 2004.

[7] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, Boca Raton, Florida, USA, 2018.

[8] Michele Colledanchise and Peter Ögren. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Transactions on Robotics*, 33(2):372–389, April 2017.

[9] Matthias Eder and Gerald Steinbauer-Wagner. A fast method for explanations of failures in optimization-based robot motion planning. In Andreas Müller and Mathias Brandstötter, editors, *Advances in Service and Industrial Robotics*, pages 114–121, Cham, 2022. Springer International Publishing. ISBN 978-3-031-04870-8.

[10] Matthias Eder, Michael Reip, and Gerald Steinbauer. Creating a robot localization monitor using particle filter and machine learning approaches. *Applied Intelligence*, 52(6):6955–6969, 2022.

[11] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 12 2003.

[12] Erann Gat, R. Peter Bonnasso, Robin Murphy, and Aaai Press. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1997.

[13] Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer, and Alexander Ferrein. Belief management for high-level robot programs. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[14] F.F. Ingrand, R. Chatila, R. Alami, and F. Robert. Prs: a high level supervision and control language for autonomous mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 43–49, 1996.

[15] Gero Iwan. History-based diagnosis templates in the framework of the situation calculus. *AI Communications*, 15(1):31–45, 2002.

[16] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*, 2010.

[17] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[18] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[19] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[20] Matteo Tadiello and Elena Troubitsyna. Verifying Safety of Behaviour Trees in Event-B. *Electronic Proceedings in Theoretical Computer Science*, 371:139–155, September 2022.

[21] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.