Evaluating Task-General Resilience Mechanisms in a Multi-Robot Team Task*

James Staley[1] and Matthias Scheutz[1][0000−0002−0064−2789]

Tufts University, Medford, MA 02155, USA
{James.Staley625703,Matthias.Scheutz}@tufts.edu

Abstract. Real-word intelligent agents must be able to detect sudden and unexpected changes to their task environment and effectively respond to those changes in order to function properly in the long term. We thus isolate a set of perturbations that agents ought to address and demonstrate how task-agnostic perturbation detection and mitigation mechanisms can be integrated into a cognitive robotic architecture. We present results from experimental evaluations of perturbation mitigation strategies in a multi-robot system that show how intelligent systems can achieve higher levels of autonomy by explicitly handling perturbations.

Keywords: Resilience · intelligent autonomous agents · long-term autonomy

1 Introduction

Resilience allows intelligent agents to accomplish their goals despite unexpected degradation of their task environment, including their own operational platform. When perturbations are possible or expected in a particular task setting, agents are typically already designed with mechanisms to cope with them (e.g., the wheels on the Mars Rovers were designed to work even when they had holes in them as it was expected that holes would eventually emerge as a result of driving over pointy rocks). While there is a large amount of literature on fault detection and recovery through software in cases where the system has a model of its operation, there are few proposals for task-general mechanisms for detecting, classifying, and mitigating perturbations.

In this paper, we discuss six dimensions of perturbations and demonstrate with a particular implementation of detection and mitigation mechanisms for a class of perturbations (those of actuation failures) how changing roles within its team allows an agent to adapt to a perturbation that cannot otherwise be mitigated. Using experimental evaluations in a simulated space-station environment of the implemented mechanisms in a multi-robot repair team, we should how detecting perturbations to a single agent's effectors and mitigating them by adapting the team member roles can enable significantly higher performance than failing to address the perturbation.

* This work was in part supported by ONR grant #N00014-18-1-2831.

2 Motivation

Perturbations happen in the real world as real task environments are not always well-behaved. A robot tasked with searching a building might be plunged into darkness when the building's power fails, for example, making it impossible to continue searching with its regular camera. The same robot's camera might fail half-way through the task, resulting in task failure. Perturbations are not limited to the environment or the robot's body, they can also occur inside the robot's computational system (e.g., an object detection algorithm could run out of memory, or have a bug that causes a crash). Some perturbations are permanent (e.g., the broken camera), while others are transient (e.g., short power outage). Some perturbations might be harmless (a door closed that was supposed to be open, that can be opened) and have no impact on the robot's task performance, or might be prohibitive (the door locked). Some could be anticipated by the robot (a storm outside might make power outages likely), while others remain unpredictable (the camera failing all of a sudden). Some perturbations can be detected by the robot (i.e., sensed or inferred such as noticing a camera must be broken because the image is frozen despite the robot's motion), while others remain undetectable (e.g., radiation that eventually destroys the computational platform). Some perturbations are avoidable (e.g., running out of battery power by charging the battery in time), while others cannot be avoided (e.g., eventually running out of memory due to increasing log files). And some perturbations can be mitigated by the robot (e.g., unlocking the locked door), while others cannot (e.g., no more memory can be added). We thus get the following six dimensions characterizing perturbations: (1) transient/permanent, (2) impactless/impactful, (3) predictable/unpredictable, (4) detectable/undetectable, (5) avoidable/unavoidable, and (6) mitigable/unmitigable.

Overall, which type of perturbation a robot encounters as part of its task performance will depend on the environment, the task, the robot's body and computational capabilities (e.g., for detecting perturbations). We would like the robot to be resilient to as many perturbations as possible, which requires it to watch out for and ideally predict perturbations in an effort to avoid them. If they are not predictable, the robot should at least detect and attempt to mitigate them, leading to the following task-independent resilience policy:

1. predict a perturbation or detect its presence
2. classify the perturbation to determine the best response
3. enact the mitigation strategy (if available)

Different refinements of this policy are possible based on the exact nature of the perturbation. For example, for a transient perturbation (e.g., light out for a short time) the best policy might be to ignore the perturbation, whereas for a permanent perturbation (e.g., camera sensor broken) the mitigation strategy might involve finding alternative ways to perform the task (e.g., tactile investigation of objects).

In the context of a multi-robot system, a perturbation must be handled with an escalating response based on the impact is has on the whole system, not just

the individual agent. In simple cases, an agent might be able to cope with the effects of the perturbation in a way that at most has a negligible impact on task performance. However, in more complex cases, a perturbation affecting a single agent might have major ramifications for the team and its organization, preventing the agent from fulfilling a critical role in the team. In such a case, the mitigation strategy cannot stop at the single agent, but must involve other agents as well, e.g., redistributing the amount and type of work each agent does.

3 Related Work

Much work on introspection and behavior modeling has focused on fault detection, diagnosis and recovery [23, 13, 19] and system reconfiguration to accomplish a high-level goal [7, 5]. Prerequisite to fault recovery is the ability to isolate a fault and determine its severity [3]. Fault detection and diagnosis methods are either analytic or data-driven (although this division becomes blurred by techniques that automate the process of traditionally analytic methods [6]). Analytic methods have a long successful history in the various engineering disciplines (e.g., the Livingstone system developed by NASA for use on Deep Space One [24]), but they are limited to tasks in constrained environments with a small number of well-understood states. Data-driven approaches employ various classification algorithms (e.g., neural networks, Bayesian networks) to learn fault models from past performance data, but this presumes the data is available (e.g., applications in aerospace can build on decades of telemetry data [14]). Golombek et al. [10, 11] attempt to find structure in temporal patterns of communications among architectural components and learn a model of normal and faulty operation by clustering those patterns. This approach is similar to our own [2] except that they can also utilize knowledge-based, top-down constraints for recovery policies.

For fault recovery (the "self-adjustment" phase in [1]), it is necessary to develop policies for how to react to faults once they have been detected. The policy can either be pre-defined or it can be learned. For example, in the Livingstone system, a recovery mechanism based on conflict resolution in a feedback controller was developed, where the control command was based on the analytic model [24]. While (possibly optimal) policies can be pre-defined or learned from existing labeled data for known faults, recovery policies for unknown faults will have to be learned dynamically, especially for systems where fault models can change during their operation.

The system employed here is able to detect anomalies faster and with less required initialization by describing an agent's actions in terms of logical primitive pre- and post-conditions. When an action systematically fails to produce its expected post-conditions it is considered to be under the effect of a perturbation. In addition, these fault detection systems typically focus on single-agent systems and so ignore the possible mitigation techniques that may be available to multi-agent systems. As a consequence, role-switching, which our experiment focuses on, has not been studied in the context of fault detection and recovery.

Multi-agent systems that are robust to perturbations have been proposed in the past. The ALLIANCE architecture designed for multi-agent fault tolerance, for example, used internal motivations and assessments to determine when an agent was underperforming at a task (potentially due to a perturbation) and should switch tasks [20, 12]. Other approaches, like Christensen et al. [4], implement an exogenous approach to fault detection, whereby member of the multi-agent group that are not perturbed can recognize reduced capabilities in their collaborators. These approaches find success in replacing a poorly performing agent, but do not investigate whether that agent may still contribute to the overall performance of the system by reassigning its role.

As we will show, role re-allocation can offer additional boosts in system performance when the agent's role explicitly depends on its dynamic capability. By first detecting that an agent's capability has changed and then assigning the compromised agent to a role where it can still contribute to the system's performance allows the multi-agent system to achieve higher performance.

4 Resilience Mechanisms and Experimental Evaluation

We are interested in evaluating architectural mechanisms for increasing an agent's resilience that are task-general and can be integrated into different agent architectures, based on the three aspects – perturbation detection, classification, and mitigation, which form a task- and hardware-independent resilience framework that cannot be found in other robotic architectures. Briefly, using introspection, an agent should monitor its task performance and compare it to expected outcomes, which allows it to recognize that action or task is systematically failing (e.g., based prior expectations of action outcomes, inferred post conditions, or other unexpected changes in the system). Specifically, through repeated monitoring and comparison of outcomes and other system states, the agent will then be able detect whether unexpected outcomes or events are simply sporadic or random, or are systematic and due to some kind of perturbation that can potentially be addressed by the agent. The overall sequence of agent actions then is: (1) model body, architectural and environmental states, as well as task performance, (2) determine whether unexpected outcomes, states or behavior, internal or external to the agent are likely noise or systematic, and in the latter case, (3) attempt to classify the kind of perturbation in terms of whether it has an impact on task performance, and if so, whether it can be mitigated. In the former case, the agent might be able to ignore it, in the latter, based on the type of perturbation, it needs to attempt to mitigate it using a number of strategies. The agents might redirect cognitive resources, change its bodily configuration, otherwise adapt to the new environment or elicit help from other agents in the context of the team.

We leverage the existing multi-agent simulation infrastructure in [8] together with a multi-agent cognitive robotic architecture (described below) to implement and evaluate a first set of perturbation detection, classification, and mitigation mechanisms and to demonstrate the utility of automatic role switching as a

possible mitigation policy in multi-robot team settings (as opposed to solely individual-based mitigation mechanisms that in some cases will fail because individual mitigation of the perturbation is not possible).

4.1 Resilience Mechanisms in the DIARC Architecture

We used the *Distributed Integrated Affect Reflection Cognition* (DIARC) architecture [22] as the basis for all algorithm developments and integration. DIARC is a component-based architecture scheme that can be instantiated for different tasks and settings with different components present in the system. It is implemented in the the "Agent Development Environment" ADE [16, 21], which, different from other robotic infrastructures such as the Robot Operating System (ROS) [**?**], was from the very beginning *designed to be as secure and fault-tolerant as possible.*[1] Hence, ADE provides the extensive architectural multi-level introspection and notification mechanisms (e.g., see [18, 17]) that can be utilized to detect and classify perturbation. System models for DIARC can either be learned offline from logged information about system states (e.g., [2]) or online as part of *anomaly detection* (e.g., [9]). DIARC's fault detection and recovery mechanisms have also been empirically evaluated in HRI studies (e.g., [15]). In the context of multi-robot teams, DIARC can be configured to run with shared architectural components that allow multiple agents to easily share information and implement distributed multi-agent architectures such as the blackboard architecture or shared mental models (e.g., [8]). We utilize this component-sharing capability to allow robots to write status and role updates to the share workspace called "Belief component" in DIARC (as it stores the agents' beliefs about themselves and the world) which allows any team member to determine what role another team member assumes and thus react to any role changes that might be necessitated by perturbations. If agents keep track of all possible roles they can assume in a team, then the Belief component could run a role assignment algorithm every time a change to the set of possible roles for a team member occurs to optimize role assignment and team performance.

In order to evaluate the effects of role-switching in response to perturbations we developed a task using the components described in this section.

We examined the beneficial effects of perturbation detection and tested the effectiveness of a multi-agent role switching mitigation strategy by performing a repair task in a simulated environment with two Willow Garage PR2 robots. The robots are simulated in ROS's Gazebo simulator, their behavior and knowledge states are governed by the DIARC architecture, and all visuals and environmental logic are controlled through Unity3D.

4.2 The Space Station Environment

We utilize an existing space station simulation environment [8] (see Fig. 1) which was designed and implemented in Unity3D. The environment consisted of three

[1] A detailed conceptual and empirical comparison of robotic infrastructures up to 2006 can be found in [16].

wings extending away from a central hub. In each wing there are two rows of 12 fuel tubes. The wings are identical but for their name designations of Alpha, Beta or Gamma. The two robots were responsible for the maintenance of fuel-tubes in an orbiting space station. Fuel-tube failures occurs regularly due to unpredictable and undetectable circumstances and all non-failing tubes have an equal chance to fail at each interval. A failed fuel-tube's condition will decay until it breaks. Once a fuel-tube has broken it cannot be repaired or fixed in any way. If enough fuel tubes break (15 for these experiments) the space station can no longer operate and must perform an emergency landing.

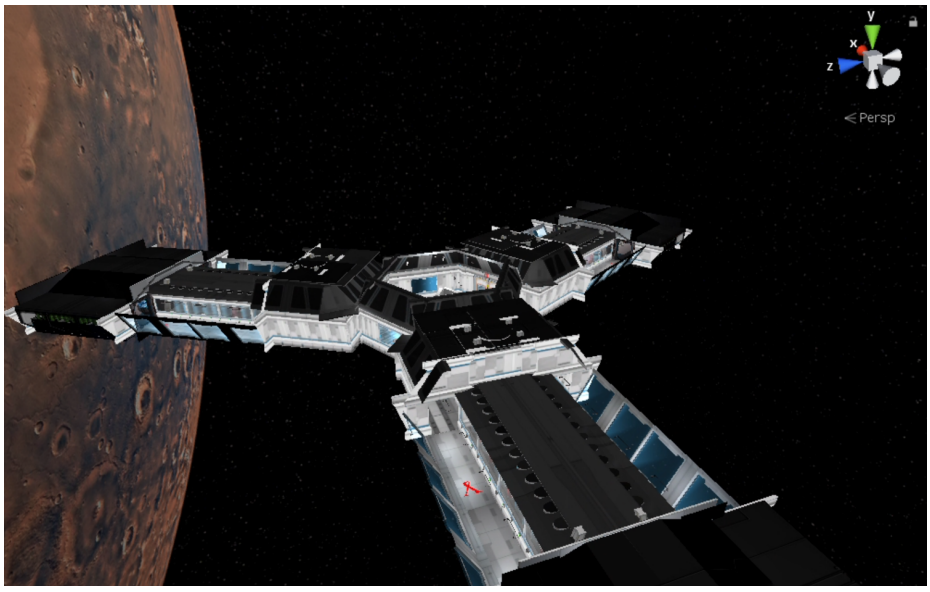

Fig. 1. Space station layout. One central hub with three marked wings.

4.3 The Robots

The simulated robots are responsible for maintaining the space station. All of the robots' planning and motion dynamics are simulated in the ROS Gazebo simulator to incorporate as realistic timing and dynamics effects on the task as possible. Robots can repair failed fuel-tubes and can perform scans that determine the condition for every fuel-tube within the wing they currently occupy.

Robots may assume one of two roles: *Repair agent* or *Sensor agent*. Repair agents move around the environment sensing and fixing failing fuel-tubes. Sensor agents cannot repair tubes, but can continue to scan wings of the space station and report their findings to the multi-agent system. The algorithms that govern these roles are described in Section 4.5.

4.4 The Search-and-Repair Task

Within this environment, two agents perform a search-and-repair task in order to keep the space installation running. Each agent initially assumes both roles of Repair and Sensor which leads to the highest task performance. We then introduce a perturbation in the form of a broken repair tool in one robot, but give the robot no way to directly sense that this perturbation has occurred. While the robot cannot determine exactly what happened, after a few failed repair attempts the goal monitoring system within DIARC architecture can infer that the error is systematic and determine that the robot can no longer repair fuel tubes. Within the DIARC action execution subsystem this means that the *repair(tube)* action no longer produces the expected post-condition of a tube in perfect (repaired) condition.

Without the ability to repair tubes, the agent's ability set has changed and it can no longer fulfill the role of Repair agent. It checks whether there is a role its ability set can still satisfy and switches to that role, in this case becoming a sole Sensor agent. The Sensor agent communicates this role change to DIARC's knowledge base and the functioning Repair agent may now use the Sensor agent's information to make better decisions for which tubes to repair (where "better decisions" in this context are heuristically implemented to keep the space station operational for longer).

4.5 Experimental Design and Procedure

Right rrom the beginning both robots patrol the space station and under normal operating conditions (i.e., both robots hold both Repair and Sensor roles) follow a basic algorithm:

1. Move to an unoccupied wing
2. Scan the wing for damaged tubes
3. Repair all damaged tubes in the currently occupied wing
4. Update the Knowledge Base

The task ends once fifteen fuel tubes have been destroyed or eighteen fuel tubes have been repaired. A maximum of eighteen is to give a hard stopping point for the case with two fully functional agents, which under ideal conditions can keep the space station operational indefinitely.

Conditions Our base condition includes two fully functioning agents who remain in perfect condition over the course of the trial. They work independently with minimal communication[2] They follow the simple algorithm described above without modification, and are expected to maintain the space station the longest.

Our two experimental conditions include one fully functioning agent and one agent with a broken welding implement, rendering the agent unable to repair

[2] Agents avoid occupying the same area of the space station.

tubes. The broken implement is a permanent, unpredictable, unavoidable, but detectable perturbation that impacts the performance of the multi-agent team. Our first experimental condition, *silent-failure*, uses an agent that is unable to detect this perturbation and continues following the simple algorithm above.

Algorithm 1: Introspective Repair Agent Role

```
Move to unoccupied wing;
Get list of damaged tubes within wing;
otherAgentIsDamaged = Check Knowledge Base for state of other agent;
if otherAgentIsDamaged and heuristic() then
    Move to damaged Sensor agent wing;
while damagedTubesInWing() do
    Go to nearest damaged tube in wing;
    Repair nearest damaged tube;
    if fixed(tube) then
        Update Knowledge Base about tube status;
    else if amDamaged() then
        Change role to Sensor agent role;
        Update Knowledge Base about own status;
        break;
end
```

Our second experimental condition, *role-switching*, uses an agent that can detect the existence of perturbation through its own reduced performance. Once this detection occurs the multi-agent team can adopt a new operating regime wherein the damaged agent drops its repair role and maintains only its role as a mobile sensor. The functioning robot, when deciding which wing of the station it should care about, can use the non-functioning robot to increase its understanding of the context and make a better decision about which action to take next. The *role-switching* team follows a new algorithm.

Algorithm 2: Introspective Sensor Agent Role

```
Move to unoccupied wing;
roles = Check Knowledge Base for own roles;
Scan wing for damaged tubes;
Update Knowledge Base about state of wing;
if !role.contains(Repair) then
    while absent(Repair Robot) do
        wait;
        Scan wing for damaged tubes;
        Update Knowledge Base about state of wing;
    end
```

The base condition and *role-switching* conditions can be described by role algorithms 1 and 2.

The agents in the base case successfully perform the Repair and Sensor roles for the duration of the task. They can scan the wing they're in and repair every damaged tube in that wing before moving on. They never enter the perturbation mitigating blocks of code because in the Repair case (1) their partner is not damaged and (2) they are not damaged, and in the Sensor case they can always hold the Repair role.

Algorithm 3: Basic Repair Agent Role

```
Move to unoccupied wing;
Get list of damaged tubes within wing;
while damagedTubesInWing() do
    Go to nearest damaged tube in wing;
    Repair nearest damaged tube;
    Update Knowledge Base about tube status;
end
```

The agents in *silent-failure* behave exactly like the base-condition agents, but to worse effect. In their case, the damaged agent has no mechanism to check whether its performance is impacted by a perturbation. Their less introspective roles can be described by algorithms 3 and 4.

Algorithm 4: Basic Sensor Agent Role

```
Move to unoccupied wing;
Scan wing for damaged tubes;
Update Knowledge Base about state of wing;
```

These basic roles perform well in the absence of perturbations, but have no ability to handle problems. We expect the multi-agent team that can *role-switch* to perform worse than the fully functioning team, because only one robot can repair tubes, but better than the *silent-failure* team, because it will cause the agent with the perturbation to attempt to repair the same tube over and over again.

4.6 Results

We ran the two robots with their respective role algorithms for $n = 17$ trials in Search-and-Repair Task. As expected, the robots in the baseline condition with no perturbation are able to keep the space station operational indefinitely as indicated by their exact average of 18 with zero standard deviation (recall,

the simulation was terminated when 18 tubes were repaired). Robots in the two perturbation conditions never made it to 18 repaired tubes (rather, 12 repaired tubes was the maximum in the role-switching condition). A two-sided Welch t-test (adjusting of unequal variances) shows a significant difference ($t(31.32) = 3.245, p = .0027$) between the silent-failure condition ($\mu = 7.58$ repaired tubes on average, $SD = 1.46$) and the resilience condition ($\mu = 9.35$ repaired tubes on average, $SD = 1.69$), as hypothesized, demonstrating that the team with perturbation detection and mitigation is able to last longer. The results thus demonstrate the utility of the introspective failure detection mechanisms that causes the agent encountering repeated action failures to give up on roles that require it to perform the failed action successfully.

5 Discussion

The multi-agent team that is able to detect the presence of a perturbation and switch roles in response outperforms the multi-agent team that can not. *Role-switching* sees numeric improvement over *silent-failure*. In certain cases the two conditions have comparable performances because the nature of this task makes the gains from a Sensor agent highly dependent on the order that the fuel-tubes break. The Sensor agent is most useful when the information it provides leads the Repair agent to a wing of the space station that it would otherwise not give immediate attention to. If this situation doesn't arise, and the Repair agent is already going to the wing most in need, then the damaged multi-agent team performs equally well with and without the *role-switching* mitigation strategy. However, we see from the data that this situation does arise often enough to justify the additional computational load required for the mitigation strategy.

Note that while both conditions perform significantly worse that the baseline condition where no perturbation occurs – as mentioned above we set the frequency of tubes breaking in such a way that two fully functional robots could about keep up with the repairs – there are cases where the resilient robots will be able to maintain an operational station while the silent failure condition will not (as a single robot will not be able to keep up with the repairs). And while this proof-of-concept evaluation is only a first indication of the utility of resilience mechanisms that adapt the agent policies in teams (to compensate for unavoidable, unrecoverable perturbations), there are many more architectural reasoning mechanisms and mitigation policies to be explored that will significantly improve the resilience of multi- agent teams.

6 Conclusion

In this paper we showed the downstream benefits of a system that can perform task-agnostic detection of perturbations and task-specific mitigation strategies to address the loss of performance associated with a given perturbation. We use a cognitive architecture, DIARC, to determine when an agent is experiencing

a systematic perturbation by detecting when an action is systematically failing to produce its expected post-conditions. Detecting the perturbation is itself extremely significant because it enables the system to respond.

We experimentally demonstrate how a multi-agent system can respond to the detected perturbation of one agent. Once an agent is damaged and cannot fulfill the repair role, the system responds by reassigning the agent to the role in which it can still contribute. The system performs better when it can perform task-specific adaptation, which it can only do because of its ability to perform task-independent perturbations detection. The ability to detect and mitigate perturbations is a requirement for an intelligent system to achieve higher levels of autonomy than what is currently possible.

References

1. Anderson, M.L., Perlis, D.R.: Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. Journal of Logic and Computation **15**(1), 21–40 (2005), `http://cogprints.org/3950/`
2. Berzan, C., Scheutz, M.: What am i doing? automatic construction of an agent's state-transition diagram through introspection. In: Proceedings of AAMAS 2012 (2012)
3. Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M., Schröder, J.: Diagnosis and Fault-Tolerant Control. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
4. Christensen, A.L., OGrady, R., Dorigo, M.: From Fireflies to Fault-Tolerant Swarms of Robots. IEEE Transactions on Evolutionary Computation **13**(4), 754–766 (Aug 2009). https://doi.org/10.1109/TEVC.2009.2017516
5. Edwards, G., Garcia, J., Tajalli, H., Popescu, D., Medvidovic, N., Gaurav, S., Petrus, B.: Architecture-driven self-adaptation and self-management in robotics systems. In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 142–151. IEEE Computer Society (2009)
6. Ernits, J., Dearden, R., Pebody, M.: Automatic fault detection and execution monitoring for auv missions. In: Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES. pp. 1–10. IEEE (2010)
7. Georgas, J.C., Taylor, R.N.: Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. pp. 105–112. SEAMS '08, ACM (2008)
8. Gervits, F., Thurston, D., Thielstrom, R., Fong, T., Pham, Q., Scheutz, M.: Toward genuine robot teammates: Improving human-robot team performance using robot shared mental models. In: Proceedings of AAMAS (2020)
9. Gizzi, E., Vie, L.L., Scheutz, M., Sarathy, V., Sinapov, J.: A generalized framework for detecting anomalies in real-time using contextual information. In: Proceedings of the 2018 IJCAI Workshop on Modeling and Reasoning in Context (MRC) (2018)
10. Golombek, R., Wrede, S., Marc, H., Martin, H.: On-line data-driven fault detection for robotic systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. San Francisco, CA (2011)
11. Golombek, R., Wrede, S., Hanheide, M., Heckmann, M.: Learning a probabilistic self-awareness model for robotic systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2010)

12. GUO, Z., YANG, W., LI, M., YI, X., CAI, Z., WANG, Y.: ALLIANCE-ROS: A Software Framework on ROS for Fault-Tolerant and Cooperative Mobile Robots. Chinese Journal of Electronics **27**(3), 467–475 (2018). https://doi.org/10.1049/cje.2018.03.001
13. Haidarian, H., Dinalankara, W., Fults, S., Wilson, S., Perlis, D., Schmill, M., Oates, T., Josyula, D., Anderson, M.: The metacognitive loop: An architecture for building robust intelligent systems. In: PAAAI Fall Symposium on Commonsense Knowledge (AAAI/CSK'10). Arlington, VA, USA (November 11-13 2010)
14. Iverson, D.L.: Inductive system health monitoring. In: International Conference on Artificial Intelligence. CSREA Press (2004)
15. Kramer, J., Scheutz, M.: Reflection and reasoning mechanisms for failure detection and recovery in a distributed robotic architecture for complex robots. In: Proceedings of the 2007 IEEE International Conference on Robotics and Automation. pp. 3699–3704. Rome, Italy (April 2007)
16. Kramer, J., Scheutz, M.: Robotic development environments for autonomous mobile robots: A survey. Autonomous Robots **22**(2), 101–132 (2007)
17. Kramer, J., Scheutz, M., Schermerhorn, P.: 'talk to me!': Enabling communication between robotic architectures and their implementing infrastructures. In: Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3044–3049. San Diego, CA (October/November 2007)
18. Krause, E., Schermerhorn, P., Scheutz, M.: Crossing boundaries: Multi-level introspection in a complex robotic architecture for automatic performance improvements. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (2012)
19. Morris, A.C.: Robotic Introspection for Exploration and Mapping of Subterranean Environments. Ph.D. thesis, Robotics Institute, Carnegie Mellon University (December 2007)
20. Parker, L.E.: ALLIANCE: an architecture for fault tolerant multirobot cooperation. IEEE Transactions on Robotics and Automation **14**(2), 220–240 (Apr 1998). https://doi.org/10.1109/70.681242
21. Scheutz, M.: ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. Applied Artificial Intelligence **20**(4-5) (2006)
22. Scheutz, M., Williams, T., Krause, E., Oosterveld, B., Sarathy, V., Frasca, T.: An overview of the distributed integrated cognition affect and reflection diarc architecture. In: Ferreira, M.A., Sequeira, J.S., Ventura, R. (eds.) Cognitive Architectures, Intelligent Systems, Control and Automation: Science and Engineering, vol. 94. Springer (2019)
23. Sykes, D., Heaven, W., Magee, J., Kramer, J.: From goals to components: a combined approach to self-management. In: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. pp. 1–8. SEAMS '08, ACM, New York, NY, USA (2008)
24. Williams, B., Nayak, P., et al.: A model-based approach to reactive self-configuring systems. In: Proceedings of the National Conference on Artificial Intelligence. pp. 971–978 (1996)