Architectural Tactics for Energy-aware Robotics Software: A Preliminary Study

Katerina Chinnappan[1], Ivano Malavolta[1], Grace A. Lewis[2], Michel Albonico[3], and Patricia Lago[1,4]

[1] Vrije Universiteit Amsterdam, The Netherlands
{i.malavolta | k.p.chinnappan | p.lago}@vu.nl
[2] Software Engineering Institute, Carnegie Mellon University, USA
glewis@sei.cmu.edu
[3] Federal University of Technology, Paraná - UTFPR, Brazil
michelalbonico@utfpr.edu.br
[4] Chalmers University of Technology, Sweden

Abstract. In software engineering, energy awareness refers to the conscious design and development of software that is able to monitor and react to energy state. Energy awareness is the key building block for energy efficiency and for other quality aspects of robotics software, such as mission completion time and safety. However, as of today, there is no guidance for practitioners and researchers on how to architect robotics software with energy awareness in mind. The goal of this paper is to identify architectural tactics for energy-aware robotics software. Specifically, using a dataset of 339,493 data points extracted from five complementary data sources (*e.g.,* source code repositories, Stack Overflow), we identified and analyzed 97 data points that considered both energy consumption and architectural concerns. We then synthesized a set of energy-aware architectural tactics via thematic analysis. In this preliminary investigation we focus on two representative architectural tactics. The identified tactics are rooted in the state of the practice in robotics software development and are potentially applicable in other application domains, such as cloud computing and the Internet of Things.

1 Introduction

Energy is a critical resource for a company's competitiveness and environmental sustainability; its management can lead to controlled operational expenses and low carbon emissions. However, data shows that energy consumption has increased considerably over time. For instance, it is projected that the industrial sector will increase energy consumption by 44% between 2006 and 2030 [13]. Furthermore, Information and Communications Technology (ICT) also plays an important role in energy consumption, where it is expected that by 2040 it will alone consume the equivalent of today's global energy production [1]. Robotics software may impact both these scenarios.

Software is becoming the prominent aspect in robotics [11]. Robotics software is becoming more and more complex in terms of control and communication, which inevitably leads to greater energy consumption [10]. Making robotics software energy-aware can lead to cost and sustainability benefits. Additionally, energy awareness is a key factor for battery-operated robots, such as autonomous cars, drones, and service robots. Being energy-aware can lead to better quality of service for the whole robotic system because the robot can operate for a longer time, more safely, and more reliably [3].

Rethinking software development is a good starting point to reason about how software systems consume energy [7]. This principle also applies to robotics software [9]. The first step towards designing energy-aware robotics software is to establish a set of concrete design options known as *architectural tactics* [4] that roboticists can use as the foundation to achieve energy awareness. In this study, we follow the definition of energy-aware software provided by Fonseca *et al.*: software that is consciously designed and developed to monitor and react to energy preferences and usage [8]. Accordingly, energy-aware tactics for robotics software can be defined as those tactics whose response is system-wide monitoring and communication of the energy levels of the robots; the monitored energy levels can be used by other components within the robotic system for different purposes, one of them (not the only one) being energy efficiency.

The goal of this paper is to *identify and document existing energy-aware tactics in state-of-the-practice robotics software*. We consider software running on the Robot Operating System (ROS) [2] as it is the de-facto standard for robotics software [10]. To achieve this goal, we build on a previously constructed dataset [9] containing online data sources specifically related to the ROS community and millions of lines of code from open-source ROS projects. The initial dataset contains 339,493 data points, which have been filtered to obtain only those data points where roboticists discuss/refer to energy consumption and discuss architecturally-relevant concerns. This filtering step produced 97 data points that were analyzed using the thematic analysis technique. As a result, we identified a set of architectural tactics for energy awareness in robotics software. In this preliminary investigation we focus on two architectural tactics, namely: (AT1) Energy Savings Mode and (AT2) Stop Current Task & Recharge. We select those tactics among the others since (i) they are among the most occurring tactics applied in the mined projects and (ii) they are complementary with respect to their objective (AT1 is about energy-level provisioning and AT2 is about mission recondiguration at runtime).

2 ROS-Based Robotics Software

ROS has become a de-facto standard that supports different robotics project domains [11]. It currently supports more than 140 types of robots [2]. Its popularity is a reflection of the vibrant ROS community, which has more than 59k questions posted on the ROS Answers forum[5], and another 2.6k discussions on ROS Dis-

[5] https://answers.ros.org

course[6]. ROS is also a framework that includes tools, libraries, and conventions for developing robotics software [2]. Its goal is to support a collaborative and open development environment, and function as middleware for robotics software, supporting the development of more complex solutions in which different high-skilled teams provide different components.

In ROS, each robotics ecosystem component (*e.g.,* robot, sensor, control application) is designed as a *node*. ROS nodes can be distributed across multiple tiers, and communicate with each other using three communication patterns: *topics*, *services*, and *actions*. *Topics* implement a publish-subscribe pattern, *i.e.,* one node publishes its data into a topic that other nodes can subscribe to and then retrieve the published data. *Services* are based on remote procedure calls (RPC), which are implemented following RPC conventions. *Actions* are used for long-running processes, where one node requests an action from another node, which starts the process, periodically publishes intermediate results, and notifies the requester node when the process is finished. These three patterns are simple and based on well-known distributed system techniques, which simplifies the development of robot communication interfaces. Furthermore, with the second version of ROS (ROS2), part of the communication relies on a Data Distribution Service (DDS) middleware, which further simplifies network programming.

3 Mining the Architectural Tactics

We extracted the architectural tactics in four distinct phases: 1) we build an initial dataset of 339,493 data points by crawling open data sources for ROS-specific data (*e.g.,* ROS Answers, Stack Overflow, GitHub); 2) we filter the dataset to extract the *562* data points that are specifically related to energy by means of a combination of keyword-based search and manual analysis; 3) we identify 97 data points where architecturally-relevant concerns are also discussed; and 4) we synthesize the architectural tactics for energy-aware robotics software via thematic analysis. Phases 1, 2, and 3 were already carried out in the context of previous work [9], where we targeted tactics for energy efficiency, *i.e.,* tactics whose response is the *reduction* of energy consumption when performing a given task. In this study, we carry out Phase 4 with a different goal, which is to identify architectural tactics for *energy awareness.*

Due to space constraints, in this paper we only provide a high-level overview of the main characteristics of two representative architectural tactics. The complete description of those tactics (including the raw data we analyzed) is available in the replication package for this study[7].

4 Results

In this section we describe in two of the tactics we extracted: (AT1) Energy Savings Mode and (AT2) Stop Current Task & Recharge. Those tactics are among

[6] https://discourse.ros.org
[7] https://github.com/S2-group/ecsa-2021-replication-package

the most occurrent ones in our dataset and their objectives are complementary. AT1 aims to provides energy-level values to other nodes in the system, a user, or a third party; differrently, AT2 update the current mission according to currently available energy, specifically by interrupting a task when the energy level becomes critical. For each tactic we provide the following information: (i) the *intent* of the tactic, (ii) the *solution* in terms of a component-and-connector and sequence diagrams that shows the main components that play a key role in the tactic and their interaction, (iii) an *example* of a concrete implementation of the tactic from our dataset.

4.1 Energy Savings Mode (AT1)

Intent. This tactic provides a shared space for storing information about the robot's state (*i.e.,* blackboard architecture pattern), which ensures that all the robotics software components are energy-aware. Thus, it informs components when they need to start saving energy and adjust their behavior accordingly.

Solution. This tactic dictates to the components in a system whether or not to enter into a state in which energy must be saved (enable or disable the energy-savings mode).

Figure 1 shows a C&C and sequence diagram for this tactic. The *Energy Savings Mode Controller* acts as a decentralized *blackboard* component [12]. Specifically, it keeps track of the current energy-savings mode of the robotic system and then makes it available to the other components in the system. In the majority of cases, the *Energy Savings Mode Controller* is proactive: it requests the current energy-savings mode from one component in the system and, based on the response, it dictates to the rest of the components to either disable or enable the energy-savings mode and change their state accordingl. Every *Observer* component receives the current energy-savings mode, switches to it, and updates the *blackboard* by sending an acknowledgment message. With this approach, all of the *Observers* are aware of the current energy-savings mode of the system, and the blackboard is aware of whether or not each component switched to the instructed energy-savings mode.

Example. The rov-control project[8] implements this tactic in a Remotely Operated Vehicle (ROV) system with two components: the *stepper* node that maps to the *Observer* component, and the *manipulator* node that maps to the *Energy Savings Mode Controller* component. The *stepper* represents nodes in the system which must comply with a certain behavior depending on the current manipulator command (*e.g.,* energy-savings mode). The *manipulator* node publishes the current command to a *manipulator_ command* topic, while the *stepper* node subscribes to it. After receiving the current *manipulator* command and complying with it, the *stepper* node publishes its new state to the *stepper_ state* topic, which is in turn subscribed to by the manipulator node.

[8] https://github.com/vortexntnu/rov-control

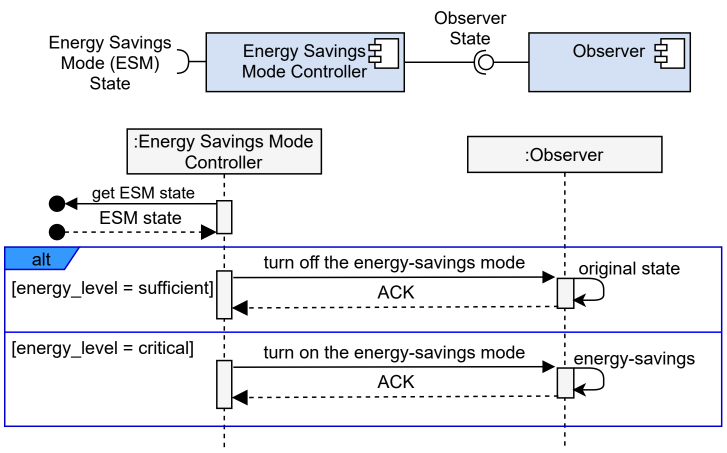
Energy Savings Mode (ESM) State
Energy Savings Mode Controller
Observer State
Observer
:Energy Savings Mode Controller
:Observer
get ESM state
ESM state
alt
[energy_level = sufficient]
turn off the energy-savings mode
ACK
original state
[energy_level = critical]
turn on the energy-savings mode
ACK
energy-savings

Fig. 1: Energy Savings Mode Tactic (AT1)

4.2 Stop Current Task & Recharge (AT2)

Intent. This tactic is used to ensure that robots are able to complete their tasks. Human intervention is not always available when the battery level is critical. Therefore, it is important that the robot is energy-aware, able to replenish its battery power when needed, and able to eventually safely complete its current mission.
Solution. This tactic gracefully interrupts a task to prevent the robot from fully discharging its battery by instructing it to recharge when the energy level reaches a critical point. The task is resumed when the battery is sufficiently charged.

Figure 2 shows the tactic components. The *Task Requestor* is responsible for requesting to execute a certain task, the *Arbiter* is responsible for deciding whether or not to stop a task and recharge the battery or execute the task, and the *Task Executor* is responsible for either stopping or executing the task. After creating a task, the *Task Requestor* sends the task to the *Arbiter*, which then checks the energy-level of the robot's battery. If the energy-level is critical, the *Arbiter* immediately removes the task and the task is not forwarded to the *Task Executor*. In the case when the energy-level is sufficient, the *Arbiter* sends the task to the *Task Executor* and the *Task Executor* starts executing the task. In parallel, the *Arbiter* starts checking the energy-level of the robot within a loop. If throughout the entire execution of the task the energy-level stays sufficient, the *Task Requestor* is notified about the completion of the task. If during the execution of the task the energy-level becomes critical, the *Arbiter* instructs the *Task Executor* to stop the task and request another component in the system to recharge the battery. Once the battery is recharged (*i.e.,* the energy level is sufficient), the *Arbiter* instructs the *Task Executor* to resume the task. Finally, when the task is completed, the *Task Executor* sends a completion message to the *Arbiter*, which in turn forwards the completion message to the *Task Requestor*.

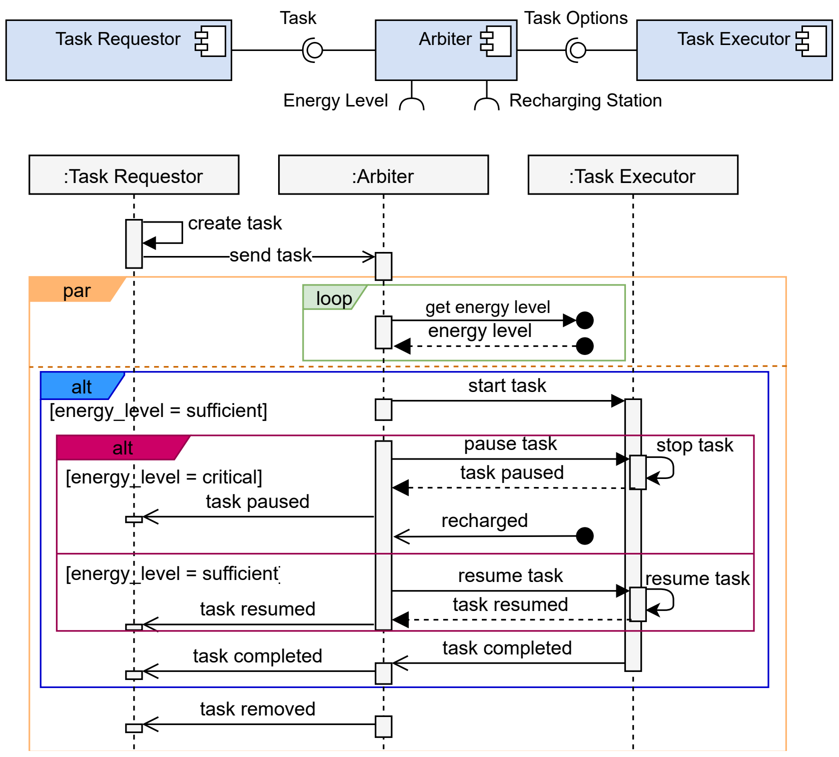

Fig. 2: Stop Current Task & Recharge Tactic (AT2)

Example. Data point *DP*24[9] implements this tactic in an autonomous multi-robot ROS-based system with two nodes: *explorer* and *exploration_planner*. In the system, the *explorer* implements the *Task Requestor* and the *Arbiter* components, while the *exploration_planner* implements the *Task Executor*. Because all the tactic decisions are made in the *explorer* (*Arbiter*), it subscribes to the *battery_state* ROS topic where the robot repeatedly publishes its battery level. Then, the *explorer* instructs the *exploration_planner* to execute an exploration task. If the battery level is not critical (set by a threshold) the *exploration_planner* node starts executing the task and reports to the *explorer* when it is finished. It is possible that during task execution the battery reaches the critical level, which is when the *explorer* saves the current progress of the task and instructs the *explorer_planner* (on the robot) to recharge the battery. The robot goes to the recharging station, and once its battery is charged, it notifies the *explorer* node, which then provides it the necessary information to continue the task.

[9] https://github.com/aau-ros/aau_multi_robot

5 Discussion

Roboticists are concerned about energy awareness. In this study, we mined ROS data sources to identify and extract architectural tactics for energy-aware robotics software. We discovered that there were in fact multiple data points that used energy-aware tactics, which indicates that there are existing methods available for roboticists to design and implement energy-aware robotics software. The tactics range from only profiling energy consumption (like in AT1) to controlling the robot to stop or pause its current task (like in AT2).
Data points are mostly for battery-operated robots, but the identified architectural tactics can be applied to other robot types. It is also interesting to note that the majority of the tactics are associated with battery-operated robots even though we did not intend to only focus on such robots. This might highlight the fact that batteries such as *Lithium Ion* and *Lithium Polymer* have a limited energy budget, and therefore battery-operated robots require an intelligent energy-management scheme. Additionally, several robotic tasks are followed by idle times where the robot needs to recharge its batteries; those idle times generally correspond to a loss of productivity [5]. Reducing idle times can lead to a better quality of service because the robot can run for a longer time and therefore more tasks can be completed. Even though most of the tactics were extracted from data points related to battery-operated robots, we argue that most of the identified tactics are applicable to robots that are powered directly from a cable (and also apply to other domains, such as cloud computing and the Internet of Things).
Roboticists tend not to explicitly document the architecture of their software. There are several benefits in documenting software architectures [6], such as helping new developers to understand projects and being able to discuss possible trade-offs. However, the analyzed data points do not present any structured documentation or diagram that model the robotics software. In this study, we provide some architecture views and tactic descriptions which may inspire roboticists to do the same as they design their software.

6 Conclusions and Future Work

In this study, we mined architectural tactics for energy-aware robotics software from data sources related to ROS-based systems. To identify energy-aware tactics in existing systems, we carried out a multi-phase study that resulted in seven *energy-awareness* tactics. To foster the applicability of the identified tactics (even beyond the ROS community), we describe them in a generic, implementation-independent manner by means of diagrams inspired by the UML component and sequence diagram notation. The presented energy-aware tactics can serve as guidance for roboticists, as well as other developers interested in architecting and implementing energy-aware software. Furthermore, the extracted energy-aware tactics can help researchers by providing empirically-grounded insights about how practitioners are designing energy-aware robotics software.

As future work, we will build a complete catalog of architectural tactics for both energy-awareness and energy-efficiency in the context of robotics software. Moreover, we are planning to conduct an empirical assessment on how different implementations of the identified tactics might impact the overall quality of robotic systems by using real robots in real missions. For example, different implementations of tactics might lead to different trade-offs with other quality attributes, such as performance and reliability.

Acknowledgments

This research is partially supported by the Dutch Research Council (NWO) through the OCENW.XS2.038 grant; and the CNPQ/FA through the PPP-CP-20/2018 call.

References

1. International Technology Roadmap for Semiconductors. Retrieved on June 04, 2021 from https://www.itrs2.net/itrs-reports.html.
2. ROS.org | Powering the world's robots. Retrieved on March 12, 2021 from https://www.ros.org/.
3. M. Albonico, I. Malavolta, G. Pinto, E. Guzmán, K. Chinnappan, and P. Lago. Mining energy-related practices in robotics software. In *Proceedings of the 18th International Conference on Mining Software Repositories, MSR*, New York, NY, May 2021. IEEE / ACM.
4. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison-Wesley Professional, 3rd edition, 2012.
5. G. Carabin, E. Wehrle, and R. Vidoni. A review on energy-saving optimization methods for robotic and automatic systems. *MDPI*, 2017.
6. P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford. Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 740–741. IEEE, 2003.
7. T. De Matteis and G. Mencagli. Proactive elasticity and energy awareness in data stream processing. *Journal of Systems and Software*, 127:302–319, May 2017.
8. A. Fonseca, R. Kazman, and P. Lago. A manifesto for energy-aware software. *IEEE Software*, 36(6):79–82, 2019.
9. I. Malavolta, K. Chinnappan, S. Swanborn, G. Lewis, and P. Lago. Mining the ROS ecosystem for green architectural tactics in robotics and an empirical evaluation. In *Proceedings of the 18th International Conference on Mining Software Repositories, MSR*, pages 300–311. ACM, May 2021.
10. I. Malavolta, G. Lewis, B. Schmerl, P. Lago, and D. Garlan. How do you architect your robots? state of the practice and guidelines for ros-based systems. In *IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 31–40, 2020.
11. I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago, and D. Garlan. Mining guidelines for architecting robotics software. *Journal of Systems and Software*, 178:110969, 2021.
12. M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline.* Prentice-Hall, 1996.

13. K. Vikhorev, R. Greenough, and N. Brown. An advanced energy management framework to promote energy awareness. *Journal of Cleaner Production*, 43:103–112, Mar. 2013.