The Usability Argument for ROS-based Robot Architectural Description Languages

Paulo Canelas [*1,2], **Bradley Schmerl** [1], **Alcides Fonseca** [2] and **Christopher S. Timperley** [1]

[1] *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, US*
[2] *LASIGE, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal*

Abstract

Component-based architectures in robotics, like the Robot Operating System (ROS), simplify system development by integrating reusable, off-the-shelf components. However, these components often lack sufficient documentation on their use and the architectures they form, leading to architectural misconfigurations. General-purpose Architectural Description Languages (ADLs) effectively provide properties to detect misconfigurations but are burdensome to use, as they require mapping new language-specific concepts to the ROS domain. This work introduces rospec, a novel ADL embedding ROS-specific knowledge to help developers specify and detect architectural misconfigurations. By mapping the language's syntax with ROS concepts, we argue that practitioners can more easily understand and reason about specifications written in the language. We provide an example and compare it with a general-purpose ADL.

Keywords: Robot Operating System. Architecture Description Language. Misconfigurations. Usability.

1 Introduction

In robot software development, component-based architectures allow developers to quickly compose and execute their systems by integrating reusable components [1]. Several such frameworks have been introduced, such as OROCOS [2], CARMEN [3], YARP [4], and the Robot Operating System (ROS) [5]. Among these, ROS has become the de facto open-source framework, with its consortium including major industry companies such as Bosch, Caterpillar, and Microsoft [6].

ROS's underlying architecture improves robotics development by allowing developers to focus on configuring and integrating reusable, off-the-shelf components [7], [8]. *Components* are processes that process inputs and may produce outputs, like nodes in a publisher-subscriber model. ROS architectures are defined at run-time through a combination of launch files, YAML parameter files, and API calls within the source code.

Nevertheless, these components often lack documentation [7], [9], [10]. Developers rely on implicit and unverified assumptions when integrating these components, which may lead to architectural misconfigurations [11], i.e., mismatched assumptions when integrating components.

Architectural Description Languages (ADLs) have been a promising venue for detecting architectural misconfigurations in various domains, including cloud computing [12], robotics [13], and cyber-physical systems [14]. In robotics, specifically, ADLs have helped developers express and understand the correct interconnection of components [15]–[18], real-time requirements [19]–[23] and hardware relationships between components [18], [19], [24].

However, an ADL's successful adoption to specific domains relies on different factors [25]. For instance, ADLs must be expressive enough to allow architects to freely describe their architecture [25], must provide domain specialization to facilitate description [26], making the syntax easy to understand [27], and maintain a proper abstraction level to match robotics engineering knowledge [26].

In this paper, we present our ongoing work on a novel architectural description language, rospec, guided by ROS domain knowledge and prior studies on ROS misconfigurations to help developers specify architectures and detect errors. We provide the usability argument that incorporating ROS domain knowledge into the ADL makes it easier for developers to understand and specify their components. We also present an example of rospec and compare it with prior work describing ROS architectures with general purpose ADLs [28].

*This work was supported by Fundação para a Ciência e Tecnologia (FCT) in the LASIGE Research Unit under the ref. (UIDB/00408/2020, UIDP/00408/2020 and EXPL/CCI-COM/1306/2021), the CMU Portugal Dual PhD program (SFRH/BD/151469/2021), and NSF-USDA-NIFA #2021-67021-33451.

PLATEAU
13th Annual Workshop at the Intersection of PL and HCI

DOI: 10.35699/1983-3652.yyyy.nnnnn

Organizers:
Sarah Chasins, Elena Glassman, and Joshua Sunshine

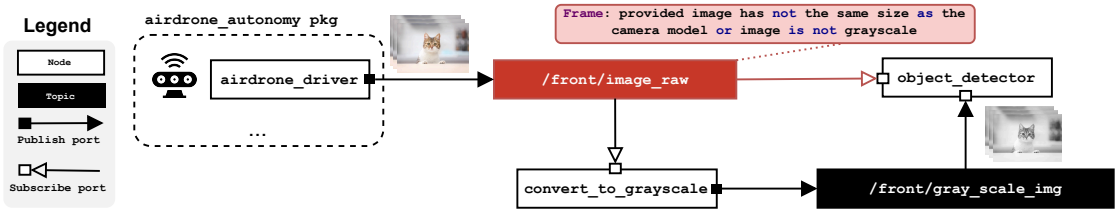

Figure 1. Example of a publisher-subscriber in ROS with a misconfiguration. The subscriber expects grayscale images, but the topic provides colored images. A possible solution is converting the colored image to grayscale.

2 Background

In this section, we provide an overview of the ROS architecture, its ecosystem of available components, integration challenges, and sources of misconfigurations.

Internally, ROS provides a loosely coupled architecture based on *nodes* and *topics* for inter-component communication. Nodes are processes that receive inputs as messages, process the data, and may produce new messages to topics – named channels for message exchange. ROS provides different communication models, such as publisher-subscriber, services, and actions, with publisher-subscriber being the model most used by developers [29]. For instance, in Figure 1, a sensor node publishes RGB camera images to a topic, `/front/image_raw`, while downstream perception nodes subscribe to this topic to receive and process these images.

ROS's key advantage is its rich ecosystem of packages comprising multiple reusable components. For example, Navigation Stack [30] provides autonomous navigation components to systems, MoveIt! [31] helps perform motion planning and manipulation, and ROS Control [32] aids with the interaction with controllers for robot actuators. Developers integrate these components into their context by changing their configurations.

Component configuration involves combining XML, YAML, and source code files and defining parameters such as topic names, hardware settings, and algorithm configurations [33]. Correctly configuring and integrating components is critical, as misconfigurations can result in dangerous behaviors.

However, documentation for components is often missing [7], [9], meaning that assumptions and patterns for good component use are hard to discover. Furthermore, the APIs do not enforce good usage, leading to misconfigurations [11]. For instance, Figure 1 presents a color format misconfiguration [11], where the subscriber assumes images are in grayscale while the publisher sends colored images. A proposed solution passes by rerouting the connection to a new node responsible for converting the colored images to grayscale.

While ROS provides a decoupled architecture for building robot software, the tradeoff gives responsibility to system developers to ensure proper component integration. Architectural misconfigurations arise from mismatched assumptions when trying to integrate different components. Although ROS supports reusability, it requires a deep understanding of the components, reading the documentation (when available), and careful configuration. Therefore, improving component descriptions is essential to help detect and prevent misconfigurations.

Despite initial efforts to describe robotics component-based architectures using general purpose ADLs [28] (e.g., Architecture Analysis & Design Language (AADL) [34]), we believe that the lack of domain specialization hinders usability, thus hindering its adoption, and motivating further work towards robotics-based ADL's to describe their architectures [35].

3 Usability of Architectural Description Languages for Robotics

In this section, we present our ongoing work on rospec, describe the architecture of Figure 1 using both AADL (Listing 1) and rospec (Listing 2), and provide three usability arguments for the benefits of integrating domain knowledge and prior studies on misconfigurations in novel ADLs for robotics.

As previously presented, Figure 1 depicts an interpretation of a color misconfiguration asked in ROS Answers, a popular Q&A Platform (question number #201031), and depicts a system that, given a

```
process airdrone_driver extends ros::node
 features
  publishes: out data port sensor_msgs::Image
  {
   ros_properties::Topic =>
        "/front/image_raw";
   ros_properties::Tag => ("COLOR", "rgb8");
  };
end airdrone_driver;

process implementation airdrone_driver.impl
 extends ros::node.impl
  subcomponents
   pub_thread: thread ros::call_pub.impl;
   connections
    pub_connection: port pub_thread.msg_out
        -> publishes;
    dac:data access parameters <->
        pub_thread.param;
   properties
    altitude_max: aadlinteger =>
        2 applies to current_component;
    altitude_min: aadlinteger =>
        1 applies to current_component;
    aadlproperty::RangeConstraint =>
        (altitude_max >= altitude_min)
        applies to current_component;
end airdrone_driver.impl;

   * * * 34 lines ommited for brevity * * *
```

Listing (1) Specification example AADL.

```
node type airdrone_driver {
  expects param altitude_max:int where {_>=0} = 2;
  expects param altitude_min:int where {_>=0} = 1;

  ensures publishes to /front/image_raw;
} where {
  altitude_max >= altitude_min;
  tag(/front/image_raw, "COLOR") == "rgb8";
}

topic type /front/image_raw : sensor_msgs/Image {
  tag(/front/image_raw, "COLOR") = "rgb8";
}

system {
  node instance driver : airdrone_driver {
    ensures param altitude_max:int where {_>=0}=2;
    ensures param altitude_min:int where {_>=0}=1;
  }

  node type object_detector {
    expects subscribes to /front/image_raw;
  } where {
    tag(/front/image_raw, "COLOR") == "gray";
  }

  node instance detector : object_detector {
    ensures subscribes to /front/image_raw;
  }
}
```

component writer

system integrator

Listing (2) Specification example rospec.

sensor, detects objects. The `airdrone_driver` node[1] publishes colored images to `/front/image_raw`, and the `object_detector` subscribes to this topic. However, the publisher node sends colored images while the subscriber expects grayscale images.

Language specialization to stakeholders improves separation of concerns and readability. In ROS, we notice two primary types of stakeholders working in the ROS ecosystem: the component writers and system integrators. *Component writers* are developers responsible for building the reusable, off-the-shelf components. These developers are contributors to the ROS ecosystem and expect the correct configuration and integration of their components to avoid spending time helping integrators with incorrect usages. *System integrators* are the developers responsible for selecting and configuring components from the ROS ecosystem to match their specific contextual and system requirement assumptions and build working robot software. Ideally, system integrators expect different components to be documented and any possible configuration errors to be raised before execution, avoiding the time-consuming task of debugging their projects, which often contain hundreds of components.

In rospec, we model these two perspectives and the concerns of the component writers and system integrators. Considering Listing 2, the first excerpt shows the component writer view of the `airdrone_driver` component, while the second one corresponds to the **`system`** integrator view. When developing their system, component writers define the interface for the constructs that exist in ROS (e.g., **`node type`** and **`topic type`**). The specifications in these interfaces can depend on configurations that enable the verification of components. System developers can interpret the documentation, document any components they create in their system, and instantiate the off-the-shelf components they use. Any clash between configurations results in an error raised during development time, shifting left the detection of misconfigurations. In contrast, when describing the architecture using AADL, the bounds between different stakeholders are not explicitly defined (e.g., through a **`system`** operator), making it more challenging to distinguish the documented component from the system descriptions.

Language specialization provides a separation of concerns, making it easier for each stakeholder to understand their role and focus, distinguish, and interpret the different aspects of the specification.

1 https://github.com/AutonomyLab/ardrone_autonomy

Embedding domain knowledge into language semantics improves understandability. When designing an ADL, domain knowledge is critical in improving developers' usability. Familiarity with the domain concepts can make it intuitive for developers to describe their system. Prior work creating ADLs motivates the need for domain-specific concepts to improve language usability [26].

When designing rospec, we considered the domain concepts regarding ROS 2 (the current generation of ROS) [33] and prior studies in ROS misconfigurations [11], [36], [37]. The embedded domain knowledge provides familiarity with concepts developers commonly use, while prior studies guide language expressiveness by using programming language (PL) concepts to detect misconfigurations.

For example, considering domain knowledge, in Listing 2, developers can create interfaces and instances related to ROS constructs (e.g., **`node`** and **`type`**), define configurable parameters using **`param`**, and describe connections between nodes and topics, using **`publishes to`** and **`subscribes to`** operators. It is expected that the semantics of a rospec connector align with those of ROS. However, when describing AADL, users remain uncertain about the semantics of a port and the additional specifications required to ensure ROS semantics are accurately represented in AADL.

When considering language expressive power to capture misconfigurations, we use well-known concepts within the programming language community, such as liquid types [38]–[41] to restrict the range of allowed values (e.g., `altitude_max : int where {_ >= 0}`, meaning positive max altitude values), and dependent types [42] to capture dependencies between different ROS constructs (e.g., `altitude_max >= altitude_min`). Moreover, we also introduce uninterpreted functions [38] to provide contextual information and semantics in the language. For instance, by using **`tag`**, component writers can provide contextual information regarding the color format when describing ROS constructs. Then, this information is checked in the **`where`** clause, allowing detection of the color misconfiguration.

On the other hand, general-purpose ADLs require the understanding and memorization of new language features so that developers can successfully describe their system. For instance, in Listing 1, developers have to define the `subcomponents` and `aadlproperty` to define constraints over the types.

Language design considering ergonomics improves ADLs to domain alignment. When designing an ADL, language ergonomics are important to reduce the friction between the language and the domain concepts. *Ergonomics*, according to Rust's language ergonomics initiative, "*is a measure of friction you experience when trying to get things done with a tool*."[2]

When designing rospec, we aimed to minimize the friction between the domain and the language by considering the following characteristics: Firstly, we identified the two distinct roles in interacting with the language and incorporated their differing perspectives into the specification. Secondly, our specification builds upon established concepts from ROS, such as parameters, topics, and nodes, which developers are already familiar with. This reduces the need to learn additional specification concepts or adapt to other semantics. Thirdly, rospec adopts the same naming conventions as ROS, ensuring consistency. For example, the node assumes that topics are published or subscribed to within the relevant context. Finally, the deployment model of rospec aligns with the ROS component distribution model, allowing each package to include its specification file.

4 Future Work

We plan to further evaluate the practicability and usability of rospec, especially considering real-world tasks in preventing misconfigurations that practitioners make. We presented two descriptions of an architecture for detecting color misconfigurations and provided a usability comparison between rospec and a general purpose ADL, AADL. In future work, we will evaluate rospec's ability to model real-world systems to detect architectural misconfigurations and perform a usability study with roboticists to understand usability issues of our language. The objective is to have an architectural description language capable of describing complex systems without sacrificing usability so that developers can adopt it to improve the ROS ecosystem.

2 https://blog.rust-lang.org/2017/03/02/lang-ergonomics.html

References

[1] H. Andreasson, G. Grisetti, T. Stoyanov, and A. Pretto, "Software architectures for mobile robots," in *Encyclopedia of Robotics*. Springer Berlin Heidelberg, 2020, pp. 1–11, ISBN: 978-3-642-41610-1. DOI: 10.1007/978-3-642-41610-1_160-1.

[2] H. Bruyninckx, "Open robot control software: The OROCOS project," in *International Conference on Robotics and Automation*, 2001, pp. 2523–2528. DOI: 10.1109/ROBOT.2001.933002.

[3] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit," in *International Conference on Intelligent Robots and Systems*, IEEE, 2003, pp. 2436–2441. DOI: 10.1109/IROS.2003.1249235.

[4] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, 2006. DOI: 10.5772/5761.

[5] M. Quigley, K. Conle, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," *ICRA Workshop on Open Source Software*, pp. 1–6, 2009.

[6] ROS-Industrial, *Current members - ros-industrial*, 2021. [Online]. Available: https://rosindustrial.org/ric/current-members/.

[7] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry, "The robot operating system: Package reuse and community dynamics," *Journal of Systems and Software*, pp. 226–242, 2019.

[8] S. Kolak, A. Afzal, M. Hilton, C. Le Goues, and C. Timperley, "It takes a village to build a robot: An empirical study of the ros ecosystem," in *International Conference on Software Maintenance and Evolution*, 2020, pp. 430–440.

[9] A. Afzal, C. Le Goues, M. Hilton, and C. Timperley, "A study on challenges of testing robotic systems," in *International Conference on Software Testing*, 2020, pp. 96–107. DOI: 10.1109/ICST46399.2020.00020.

[10] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: A perspective from the service robotics domain," in *ESEC/FSE European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 593–604. DOI: 10.1145/3368089.3409743.

[11] P. Canelas, B. Schmerl, A. Fonseca, and C. S. Timperley, "Understanding misconfigurations in ros: An empirical study and current approaches," in *International Symposium on Software Testing and Analysis*, 2024. DOI: 10.1145/3650212.3680350.

[12] A. Bergmayr, U. Breitenbücher, N. Ferry, *et al.*, "A systematic review of cloud modeling languages," *ACM Computing Surveys*, 22:1–22:38, 2018. DOI: 10.1145/3150227.

[13] A. Nordmann, N. Hochgeschwender, D. L. Wigand, and S. Wrede, "A survey on domain-specific modeling and languages in robotics," *Journal of Software Engineering in Robotics*, pp. 75–99, 2016.

[14] K. Wan, K. L. Man, and D. Hughes, "Specification, analyzing challenges and approaches for cyber-physical systems (cps)," *Engineering Letters*, 2010.

[15] N. Gobillot, C. Lesire, and D. Doose, "A modeling framework for software architecture specification and validation," in *Simulation, Modeling, and Programming for Autonomous Robots*, 2014, pp. 303–314. DOI: 10.1007/978-3-319-11900-7_26.

[16] H. Muccini and M. Sharaf, "Caps: Architecture description of situational aware cyber physical systems," in *International Conference on Software Architecture*, 2017, pp. 211–220. DOI: 10.1109/ICSA.2017.21.

[17] F. J. Ortiz, D. Alonso, F. Rosique, F. Sánchez-Ledesma, and J. A. Pastor, "A component-based meta-model and framework in the model driven toolchain c-forge," in *Simulation, Modeling, and Programming for Autonomous Robots*, 2014, pp. 340–351. DOI: 10.1007/978-3-319-11900-7_29.

[18] H. Wei, X. Duan, S. Li, G. Tong, and T. Wang, "A component based design framework for robot software architecture," in *International Conference on Intelligent Robots and Systems*, 2009, pp. 3429–3434. DOI: 10.1109/IROS.2009.5354161.

[19] M. Frigerio, J. Buchli, and D. G. Caldwell, *A domain specific language for kinematic models and fast implementations of robot dynamics algorithms*, 2013. arXiv: 1301.7190 [cs.RO].

[20] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "Robotml, a domain-specific language to design, simulate and deploy robotic applications," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2012, pp. 149–160, ISBN: 9783642343261. DOI: 10.1007/978-3-642-34327-8_16.

[21] M. Look, A. Navarro Perez, J. O. Ringert, B. Rumpe, and A. Wortmann, *Black-box integration of heterogeneous modeling languages for cyber-physical systems*, 2014. arXiv: 1409.2388 [cs.SE].

[22] A. Rajhans, S.-W. Cheng, B. Schmerl, *et al.*, "An architectural approach to the design and analysis of cyber-physical systems," *Electronic Communication of the European Association of Software Science and Technology*, 2009. DOI: 10.14279/tuj.eceasst.21.286.

[23] J. Martinez, A. Ruiz, A. Radermacher, and S. Tonetta, "Assumptions and guarantees for composable models in papyrus for robotics," in *International Workshop on Robotics Software Engineering*, 2021, pp. 1–4. DOI: 10.1109/RoSE52553.2021.00007.

[24] A. Singh, F. Quint, P. Bertram, and M. Ruskowski, "A framework for semantic description and interoperability across cyber-physical systems," *International Journal on Advances in Intelligent Systems*, pp. 70–81, 2019.

[25] S. Hussain, "Investigating architecture description languages (adls) a systematic literature review," M.S. thesis, Software and Systems, The Institute of Technology, 2013, p. 56.

[26] E. Woods and R. Hilliard, "Architecture description languages in practice session report," in *Working Conference on Software Architecture*, 2005, pp. 243–246. DOI: 10.1109/WICSA.2005.15.

[27] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *Transactions of Software Engineering*, no. 1, pp. 70–93, 2000. DOI: 10.1109/32.825767.

[28] G. Bardaro, A. Semprebon, A. Chiatti, and M. Matteucci, "From models to software through automatic transformations: An AADL to ROS end-to-end toolchain," in *International Conference on Robotic Computing*, 2019, pp. 580–585. DOI: 10.1109/IRC.2019.00118.

[29] A. Santos, A. Cunha, N. Macedo, R. Arrais, and F. N. dos Santos, "Mining the usage patterns of ROS primitives," in *International Conference on Intelligent Robots and Systems*, 2017, pp. 3855–3860. DOI: 10.1109/IROS.2017.8206237.

[30] D. Coleman, I. A. ucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *Journal of Software Engineering for Robotics*, pp. 3–16, 2014. [Online]. Available: http://arxiv.org/abs/1404.3785.

[31] S. Chitta, I. A. Sucan, and S. Cousins, "Moveit! [ROS topics]," *Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[32] S. Chitta, E. Marder-Eppstein, W. Meeussen, *et al.*, "ros_control: A generic and simple control framework for ROS," *The Journal of Open Source Software*, vol. 2, p. 456, 2017. DOI: 10.21105/JOSS.00456.

[33] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, 2022. DOI: 10.1126/scirobotics.abm6074.

[34] P. H. Feiler, D. P. Gluch, and J. Hudak, "The architecture analysis & design language (aadl): An introduction," 2006.

[35] C. Le Goues, S. G. Elbaum, D. J. Anthony, *et al.*, "Software engineering for robotics: Future research directions; report from the 2023 workshop on software engineering for robotics," *CoRR*, vol. abs/2401.12317, 2024. DOI: 10.48550/ARXIV.2401.12317.

[36] N. Albergo, V. Rathi, and J. Ore, "Understanding xacro misunderstandings," in *International Conference on Robotics and Automation*, 2022, pp. 6247–6252. DOI: 10.1109/ICRA46639.2022.9812349.

[37] S. Kate, J. Ore, X. Zhang, S. G. Elbaum, and Z. Xu, "Phys: Probabilistic physical unit assignment and inconsistency detection," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 563–573.

[38] P. M. Rondon, M. Kawaguchi, and R. Jhala, "Liquid types," in *Conference on Programming Language Design and Implementation*, ACM, 2008, pp. 159–169. DOI: 10.1145/1375581.1375602.

[39] C. Gamboa, P. Canelas, C. S. Timperley, and A. Fonseca, "Usability-oriented design of liquid types for java," in *International Conference on Software Engineering*, 2023, pp. 1520–1532. DOI: 10.1109/ICSE48619.2023.00132.

[40] N. Lehmann, A. T. Geller, N. Vazou, and R. Jhala, "Flux: Liquid types for rust," *Proceedings of the ACM Programming Languages*, vol. 7, no. PLDI, pp. 1533–1557, 2023. [Online]. Available: https://doi.org/10.1145/3591283.

[41] N. Vazou, "Liquid haskell: Haskell as a theorem prover," Ph.D. dissertation, University of California, San Diego, USA, 2016.

[42] H. Xi and F. Pfenning, "Dependent types in practical programming," in *Symposium on Principles of Programming Languages*, A. W. Appel and A. Aiken, Eds., 1999, pp. 214–227. DOI: 10.1145/292540.292560.