Architectural Model Inference from Code for ROS-based Robotics Systems

Tobias Dürschmid, Christopher S. Timperley, David Garlan, and Claire Le Goues
Carnegie Mellon University, Pittsburgh, PA, USA
{tdurschm, ctimperl, garlan, clegoues}@cs.cmu.edu

***Abstract*—Model-based analysis is a common technique to identify incorrect behavioral composition of complex, safety-critical systems, such as robotics systems. However, creating structural and behavioral models for hundreds of software components manually is often a labor-intensive and error-prone process. In this paper, we present our past, current, and ongoing work to infer structural and behavioral models for components of systems based on the Robot Operating System (ROS) using static analysis by exploiting assumptions about the usage of the ROS framework. We see this work as a contribution towards making well-proven and powerful but infrequently used methods of model-based analysis more accessible and economical in practice to make robotics systems more reliable and safe.**

I. Introduction

Robotics systems, especially systems written for the Robot Operating System (ROS) [1], are often *component-based*: They are implemented as independently deployable run-time units that communicate with each other primarily via messages [1]–[5]. The composition and evolution of software components is error prone, due to undocumented assumptions that might change over time. When composed inconsistently, the behavior of these systems can be unexpected, such as a component indefinitely waiting, not changing to the desired state, ignoring inputs, message loss, or publishing messages at an unexpectedly high frequency [6]–[8].

Software architects commonly use model-based architecture analysis to ensure the safety and correct composition of components [9]–[16]. Based on structural and behavioral models, such as state machines, of the current system, architects can find inconsistencies or predict the impact of changes on the system's behavior. However, in practice, due to the complexity of robotics systems, creating models manually is time-consuming and difficult [9], [13], [17]. This motivates work on automated model recovery to reduce the modeling effort and make formal analysis more accessible in practice.

To address the challenge of automatically inferring behavioral component models for ROS-based systems, we propose to use static analysis of the system's source code written in C++. In general, inferring behavioral models statically is undecidable [18]. Even a partial solution is practically challenging, because the analysis needs to infer what subset of arbitrary C++ code gets compiled to be executed as a single component, what subset of this component's code communicates with other components, and under what situations this code for inter-component-communication is reachable. Fortunately, the following observations about the ROS ecosystem make this problem tractable for most cases in practice:

1) Component architectures and behaviors are defined via Application Programming Interface (API) calls that have well-understood architectural semantics [19].
2) The composition and configuration of components to build larger systems is done in separate architecture configuration files (i.e., launch files). Most of these result in "quasi-static" systems. That is, architectures rarely change following run-time initialization [19].
3) Behavioral patterns, such as periodically sending messages, are usually implemented using features provided by the ROS framework. Hence, most instances of those patterns follow a similar implementation template.

Based on these observations, we present the following contributions:

1) An approach and open-source implementation to statically infer component-connector models and component behavioral models from ROS code.
2) The first available data set of 29 architecture misconfiguration bugs across 5 open-source ROS system and 106 component models of the Autoware system that we used to evaluate our approach.
3) Our ongoing and future work to combine static analysis with dynamic analysis to add timing information and resolve known unknown in the models inferred statically.

II. Inferring Component-Connector Models

Our existing tool ROSDiscover [5] statically recovers component-connector models to obtain *structural models* of individual ROS components from their C++ source code. Each model structural description of a given node in terms of its interface (i.e., topics, services, and actions).

First, ROSDiscover recovers a parametric component interface models containing the component's ports by looking for API calls that define subscribers, publishers, actions, or services and uses symbolic execution to resolve the possible values of API call arguments. Then, it connects individual component interface models to full-system component-connector models by analyzing the launch files that connect the ports of component and instantiates parameter of the interface models. Finally, it finds architecture reconfiguration bugs by checking well-formedness rules specified in first-order logic on the system model.

In our evaluation on five complex real-world open-source ROS systems, we found that ROSDiscover's recovery of component interface models has an accuracy of 85 %, recovery of component-connector models has an accuracy of 90 % [5].

III. Inferring Component Behavioral Models

While ROSDiscover can recover structural models, it does not reconstruct *component behavior*, i.e., dynamic aspects that describe how the component reacts to inputs and how it produces outputs, such as whether a component sends a message in response to receiving an input, whether it sends messages periodically or sporadically, and what state conditions or inputs determine whether it sends a message. Therefore we developed an extension, called `ROSInfer` that statically infers reactive, periodic, and state-based behavior of ROS components to create a state machine of architecturally-relevant behavior.

Similar to recovering structural models, we can also made the observation that the ROS API is commonly used to implement architecturally-relevant behavior. By looking for the API calls that define callbacks for receiving a message (`ros::NodleHandle::subscribe`), sending a message (`ros::Publisher::publish`), or sleeping for the remaining time of a periodic interval (`ros::Rate::sleep`), we recover models of architecturally-relevant behavior that can then be used for model-based analysis of the system. `ROSInfer` reconstructs state machine models by identifying ROS API calls that implement these types of behavior, their argument values, and the control flow between them.

We recover reactive behavior by finding control flow from a subscriber callback to a publish call. This establishes causality between receiving a message and sending another message.

To recovery periodic behavior, `ROSInfer` looks for publish calls within loops that have infinite conditions (`true` or `ros::ok`) that call sleep on a rate object. Recovering the frequency defined in the rate constructor tells lets us recover the target frequency of the periodic behavior.

To cover state-depended behavior, `ROSInfer` finds state variables, their initial values, and state transitions. Our heuristics to identify state variables are (1) the variable is used in control conditions of architecturally-relevant behavior (i.e., functions that send messages, functions that change state variables, and of their transitive callers) and (2) the variable is in global or component-wide scope, such as member variables of component classes or non-local variables. To infer the initial state (i.e., the initial values for each state variable) of the component, `ROSInfer` searches for the first definitions of the variables either in their declaration or the main method. After the state variables are identified, `ROSInfer` infers transition conditions by combining control conditions of architecturally relevant behavior using logical operators `and` and `not` depending on whether the path is taking a negation branch (e.g., the else branch of an if-statement).

We evaluated `ROSInfer` on 106 components of Autoware,[1] the world's leading open source autonomous driving software, by comparing the recovered behavior with a ground-truth obtained by manually inspecting the code and creating hand-written models of their actual behavior. If a behavior was not found or a value not recovered, we traces this false negative back to limitations of the implementation that can be fixed with more engineering effort or limitations of the approach. We find that on our data set, the approach could recover 100 % of periodic behaviors, 84 % of reactive behaviors, 55 % of state variables, and 67 % of state transitions.

[1] https://www.autoware.org

IV. Combination of Static and Dynamic Analysis

As the results from our evaluation of our current work have shown even perfect static analysis still leaves incomplete models in some cases. Furthermore, static analysis cannot infer execution times of tasks, producing models that cannot be used for most kinds of performance analysis, bottle neck analysis, or analysis of race condition. Fortunately, since the models are directly derived from the source code, they could also be used to guide the creation of experiments for dynamic analysis to fill in the unknown values in incomplete models, or to identify representative paths through the system that be used for profiling. This motivates future work on combining static and automated dynamic analysis to infer behavioral component models that contain more information about the components.

We are currently planning to extent `ROSInfer` with dynamic analysis that automatically deploys components, systematically sends messages to it based on the known state machines to collect timing data or to resolve known unknowns.

V. Possible Analyses for Inferred Models

Combining behavioral component models with component-port-connector models, allows for analyses of intra-component-data-flow. Structural models alone do not contain information how the inputs of a component are used and what is needed for the component to produce an output. Having input-output state machine models like the ones `ROSInfer` infers, allows to trace which messages at one component cause messages to be sent in other parts of the system. To check whether the components of a system are composed correctly, properties such as "An input at input port I_1 of component C_a can/must result in an output at output port O_1 of C_b" can be checked via discrete event simulation [20] or logical reasoning [21].

Furthermore, synchronizing the resulting component state machines at their input/output messages allows for checking arbitrary Linear Temporal Logic (LTL) properties via approaches such as PRISM [22]. Thereby safety and security properties, such as the component changing to a desired state, no messages getting lost or ignored, or a component eventually publish a certain message, can be checked [23]–[25].

Additionally, knowing the frequencies at which periodic messages get published allows to propagate these frequencies to all transitive receivers of this data stream so allow to check the desired frequency of message publishing further down the data stream to avoid unexpectedly high publishing frequencies.

REFERENCES

[1] M. Quigley, "Ros: An open-source robot operating system," in *International Conference on Robotics and Automation Workshop on Open Source Software*, 2009, [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/BinPicking/2016_RodrigoSalgueiro/LIB/ROS/icraoss09-ROS.pdf.

[2] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016, DOI: https://doi.org/10.1016/j.jss.2016.08.039.

[3] M. Y. Jung, A. Deguet, and P. Kazanzides, "A component-based architecture for flexible integration of robotic systems," in *International Conference on Intelligent Robots and Systems*, 2010, pp. 6107–6112, DOI: 10.1109/IROS.2010.5652394.

[4] D. Brugali, A. Brooks, A. Cowley, *et al.*, "Trends in Component-Based Robotics," in *Software Engineering for Experimental Robotics*. Springer Berlin Heidelberg, 2007, pp. 135–142, DOI: 10.1007/978-3-540-68951-5_8.

[5] C. S. Timperley, T. Dürschmid, B. Schmerl, D. Garlan, and C. Le Goues, "Rosdiscover: Statically detecting run-time architecture misconfigurations in robotics systems," in *Proceedings of the 19th IEEE International Conference on Software Architecture.*, ser. ICSA '22, IEEE, 2022, pp. 112–123, DOI: 10.1109/ICSA53651.2022.00019.

[6] R. Halder, J. Proença, N. Macedo, and A. Santos, "Formal Verification of ROS-Based Robotic Applications Using Timed-Automata," in *International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, 2017, pp. 44–50, DOI: 10.1109/FormaliSE.2017.9.

[7] P. Canelas, M. Tavares, R. Cordeiro, A. Fonseca, and C. S. Timperley, "An Experience Report on Challenges in Learning the Robot Operating System," in *International Workshop on Robotics Software Engineering (RoSE)*, 2022, pp. 33–38, DOI: 10.1145/3526071.3527521.

[8] C. S. Timperley, G. van der Hoorn, A. Santos, H. Deshpande, and A. Wasowski, "ROBUST: 221 Bugs in the Robot Operating System,"

[9] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, p. 101 021, 2021, DOI: https://doi.org/10.1016/j.cola.2020.101021.

[10] D. Brugali, "Model-Driven Software Engineering in Robotics," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 155–166, 2015, DOI: 10.1109/MRA.2015.2452201.

[11] C. Newcombe, "Why Amazon Chose TLA+," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, Springer Berlin Heidelberg, 2014, pp. 25–39, DOI: 10.1007/978-3-662-43652-3_3.

[12] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How amazon web services uses formal methods," *Commun. ACM*, vol. 58, no. 4, pp. 66–73, Mar. 2015, DOI: 10.1145/2699417.

[13] M. Weißmann, S. Bedenk, C. Buckl, and A. Knoll, "Model Checking Industrial Robot Systems," in *Model Checking Software*, Springer Berlin Heidelberg, 2011, pp. 161–176, DOI: 10.1007/978-3-642-22306-8_11.

[14] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal Specification and Verification of Autonomous Robotic Systems: A Survey," *ACM Comput. Surv.*, vol. 52, no. 5, Sep. 2019, DOI: 10.1145/3342355.

[15] H. Araujo, M. R. Mousavi, and M. Varshosaz, "Testing, Validation, and Verification of Robotic and Autonomous Systems: A Systematic Review," *ACM Trans. Softw. Eng. Methodol.*, Jun. 2022, Just Accepted, DOI: 10.1145/3542945.

[16] F. Ingrand, "Recent Trends in Formal Validation and Verification of Autonomous Robots Software," in *International Conference on Robotic Computing (IRC)*, 2019, pp. 321–328, DOI: 10.1109/IRC.2019.00059.

[17] M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, "Automatic Modeling and Simulation of Robot Program Behavior in Integrated Virtual Preparation and Commissioning," *Procedia Manufacturing*, vol. 11, pp. 284–291, 2017, International Conference on Flexible Automation and Intelligent Manufacturing, DOI: https://doi.org/10.1016/j.promfg.2017.07.107.

[18] W. Landi, "Undecidability of Static Analysis," *ACM Lett. Program. Lang. Syst.*, vol. 1, no. 4, pp. 323–337, Dec. 1992, DOI: 10.1145/161494.161501.

[19] 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 (IROS '17)*, IEEE, 2017, pp. 3855–3860, DOI: 10.1109/IROS.2017.8206237.

[20] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-Based Reliability Prediction with the Palladio Component Model," *IEEE Transactions on Software Engineering (TSE)*, vol. 38, no. 6, pp. 1319–1339, Nov. 2012, DOI: 10.1109/TSE.2011.94.

[21] M. Kmimech, M. T. Bhiri, and P. Aniorte, "Checking component assembly in acme: An approach applied on uml 2.0 components model," in *2009 Fourth International Conference on Software Engineering Advances*, 2009, pp. 494–499, DOI: 10.1109/ICSEA.2009.78.

[22] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *Computer Performance Evaluation: Modelling Techniques and Tools*, Springer Berlin Heidelberg, 2002, pp. 200–204.

[23] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi, "Systematic model-based safety assessment via probabilistic model checking," in *Leveraging Applications of Formal Methods, Verification, and Validation*, Springer Berlin Heidelberg, 2010, pp. 625–639.

[24] X. Ge, R. F. Paige, and J. A. McDermid, "Analysing system failure behaviours with prism," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion*, 2010, pp. 130–136, DOI: 10.1109/SSIRI-C.2010.32.

[25] N. Alexiou, S. Basagiannis, and S. Petridou, "Formal security analysis of near field communication using model checking," *Computers & Security*, vol. 60, pp. 1–14, 2016, DOI: https://doi.org/10.1016/j.cose.2016.03.002.