Developing Robot Software Models using LLMs and Formal Verification

Richard North[1], Luke Jackson[1],
Ipek Caliskanelli[1,2], Ana Cavalcanti[1], and Pedro Ribeiro[1]

[1] Department of Computer Science, University of York
[2] UKAEA

Abstract. In this paper, we introduce an approach for developing software models for robots using a domain-specific language: RoboChart. Our proposed approach leverages Large Language Models, with the support of formal verification, to reduce the effort of creating RoboChart models. With (verified) RoboChart models, we can obtain code and tests for the software. We present our vision and preliminary results.

1 Introduction

The application of Large Language Models (LLMs) in coding reveals significant potential for utilising them in innovative ways [4]. Here, we propose its joint use with formal verification for developing design models for robot software.

Designing a robot is a complex and time-consuming task. The RoboStar framework provides a comprehensive model-based approach to design and verify software for robots [3]. It supports code generation, and verification by simulation, testing, and proof, catering for requirements of the robotic platform and of human interaction. At the core of RoboStar is RoboChart [9], a diagrammatic Domain-Specific Language (DSL), for modelling and verifying the functional, timed, and probabilistic behaviours of robotic software.

A RoboChart model is defined by a module: a robotic platform and one or more controllers, defined by one or more state machines. The platform defines events, operations, and variables representing *services required from the robot* for use by the software. Distinctively, they can specify *timing properties and probabilistic choices*. They also have a formal semantics for refinement.

RoboChart models insulate roboticists from the mathematical underpinnings of the RoboStar techniques, using familiar concepts to capture requirements. Yet, the definition of models is challenging, and our approach provides assistance to develop validated models, minimising errors in design due to human oversight.

In our proposed approach, requirements are translated into a formal property language, specifically RoboCert [8, 13] using an LLM. As a readable, but formal notation, RoboCert enables iterative improvements to natural language requirements using an unambiguous account as a target. Our proposal is to use RoboCert requirements to guide an(other) LLM to define a RoboChart model. After we have such a model, we check for syntax correctness, well-formedness,

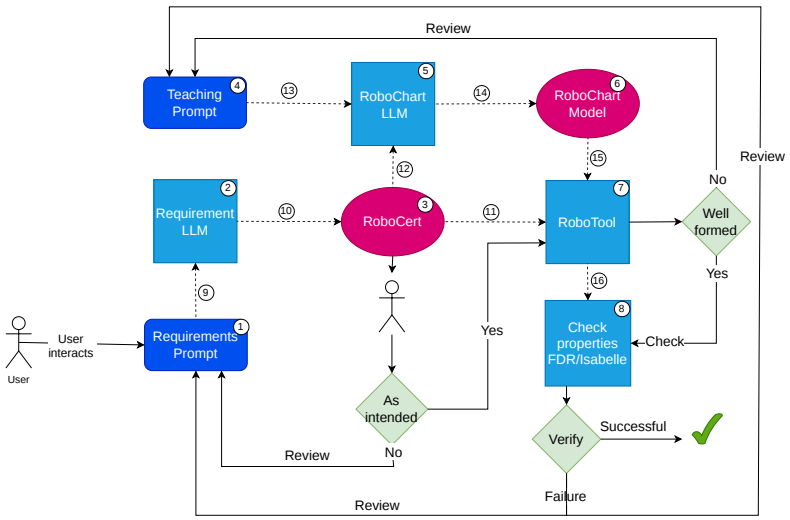

Fig. 1. Approach: dashed arrows indicate data flow, and solid arrows, workflow

and then finally semantic correctness by comparing its formal semantics to that of the RoboCert requirements. In this approach, the requirements, as captured in natural language, RoboCert, and finally RoboChart, play a central role.

In summary, the work we propose utilises two LLMs. The first generates RoboCert requirements, while the second produces RoboChart models based on those specifications. This process incorporates learned patterns provided by a teaching prompt, allowing us to subsequently test and validate the resulting models. Our preliminary results so far, however, cover the generation of RoboChart models directly from natural language requirements.

2 LLM-based generation of RoboChart models

In this section, we describe our proposed approach and preliminary results.

Overall approach. Our vision is depicted in Figure 1. Our approach uses an existing teaching prompt((4) in Figure 1) as input to an LLM (5) that creates RoboChart models based on requirements of the software (1). These requirements are described using a RoboStar language called RoboCert (3), which mixes sequence diagrams and controlled natural English (CNL) [12, 13]. To create the RoboCert document, an(other) LLM (2) uses a prompt that describes the requirements in English. This means that the requirements prompt that is in natural language (1) is passed to the LLM (2) and RoboCert (3) is the output which is then checked to ensure that it is as intended. If it is not, the requirements prompt is revised. Otherwise, the RoboCert requirements can be used as input for later verification of RoboChart models.

The point of using RoboCert is to describe the requirements in a precise way, using a notation that has a formal semantics that can be used to verify

RoboChart models. Because RoboCert documents are readable, it is feasible for the user to check the descriptions generated by the requirements LLM. (There is also the potential for formal analysis of RoboCert requirements.) This, combined with the various notations we utilise for RoboCert—such as controlled natural language for refinement properties and for probabilistic temporal logic (for use in model checking), and sequence diagrams—has the potential to enhance the overall results in the generated RoboChart models.

The validity of the RoboChart model, that is, syntactic correctness and well formedness, can be checked using the RoboChart modelling and verification tool, called RoboTool (7)[3]. If RoboTool flags errors of this nature, this feedback is integrated back into the teaching prompt (4), and the LLM is reprompted (5). If the RoboChart model is valid, it is submitted to a model checker or theorem prover (8). RoboTool can generate automatically a CSP [5] script and an Isabelle [11] theory, which can be used to ensure that the RoboChart model meets the RoboCert requirements. If the verification succeeds, the process is complete. If not, the RoboChart model and a counterexample generated by FDR or Isabelle are used to enrich the requirements prompt, if the issue is a problem with the requirements, or the teaching prompt, if the issue is with RoboChart.

Of course, there is also the possibility of fixing the RoboChart model by hand. This will depend on an evaluation of whether the model obtained is close enough to that envisaged. Judging the relative effort of reprompting or modifying the model can be helped by analysis of the counterexamples provided by verification.

The quality of the RoboChart models that can be obtained depends on the LLMs used. Even if LLMs evolve to consistently provide excellent results, however, the use of our approach is still of relevance. It provides the evidence of quality that will be missed if we just accept those results.

Below, we describe preliminary results and experiments towards our vision.

Teaching prompts. The teaching prompt is structured as a sequence of examples of RoboChart models. For each example, we have three parts. The first part gives a natural language description of the requirements used to develop the RoboChart model. The second part is the model itself (provided in a textual format accepted by RoboTool). The third part is a natural language description of the model written by hand: it defines the RoboChart robotic platform, therefore, the inputs and outputs, the controllers, and the machines, in detail.

For evaluation, we have considered four RoboChart models for: the alpha algorithm [2], a chemical detector robot [6], and two vacuum cleaner robots [13]. These four RoboChart models have been incorporated in the teaching prompt ((4) in Figure 1) so that the LLM is supplied with concrete examples of complete RoboChart models to enhance syntax and well-formedness in generated models. The teaching prompt and all artefacts of our experiment are available[4].

Qualitative results. We have then evaluated the approach by constructing a natural language description of a previously unseen robotic system, and asking

[3] `https://robostar.cs.york.ac.uk/robotool/`

[4] https://github.com/UoY-RoboStar/robochart-llm-gen/

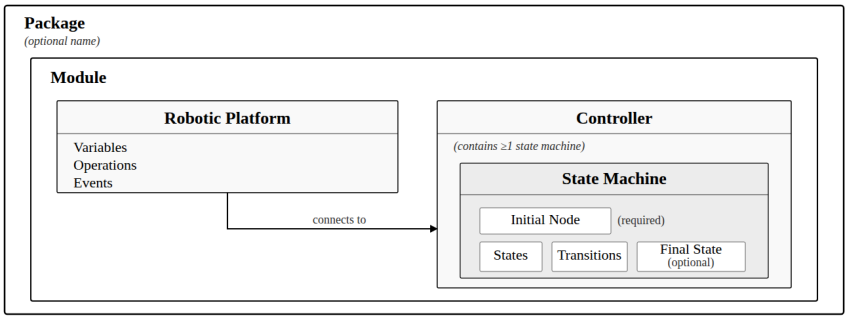

Fig. 2. The minimal valid RoboChart structure

for a corresponding RoboChart model using seven LLMs: GPT 4.1[5] and o3, Gemini 2.5 Flash and Pro[6], Gemma3 27B, DeepSeek-R1 70B and 671B.

As a small example, we present the following natural language description of the requirements for the walking nose robot used to generate a RoboChart model. "*The WalkingNose Robot performs a random walk. While it moves, it senses for gases in the environment. If gas is detected at a concentration above some threshold, a flag event is raised and execution stops.*" The RoboChart model generated by the LLM effectively identifies the system's requirements, establishing the need for one module, one platform, a single controller, and a state machine with a plausible structure and control flow.

The minimal valid structure for a RoboChart model is described in Figure 2. Impressively, the description of a RoboChart model that follows this structure was accomplished in a few-shot context, that is, where the teaching prompt includes examples. In particular, the definition of the abstraction of the services of the robotic platform is not trivial. This highlights the LLM's capabilities, as they were achieved through straightforward prompting rather than model fine-tuning or techniques like RAG.

The experimentation indicates that pre-trained models, in particular GPT 4.1 and Gemini 2.5 Pro, can successfully generate syntactically correct RoboChart models most of the time, although they can make minor syntactic errors. Gemini 2.5 Flash generates models with more severe syntax errors, for example, using operations instead of events even when explicitly asked to use an event. For outputs, events or operations can normally be used, but the choice has an impact on how components can be connected, and can create problems. Smaller LLMs like Gemma3 27B and DeepSeek-R1 70B fail to generate RoboChart models with a sensible structure or correct syntax, and are much more prone to hallucination.

In general, the generated models can sometimes be ill-formed, for example, violating RoboChart scoping rules by not declaring variables or events. We have

[5] `https://openai.com/index/gpt-4-1/`

[6] `https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-model-thinking-updates-march-2025/`

also experimented with augmenting the prompts with well-formedness rules from RoboChart's reference manual to obtain improvements in output quality. This has eliminated problems in the models generated, and further experiments will include in the teaching prompt a full account of the well-formedness conditions.

Our experiments so far have not covered the use of RoboCert. We expect, however, that the generation of RoboChart models and RoboCert documents will inevitably face similar challenges in that they both are formal and are being created from informal requirements. Because of this there is a need for review of these documents and models to ensure that they encapsulate the requirements.

3 Future and related work

In conclusion, LLMs have the potential to shape the future of how we generate models of robot software. Future work will experiment further with our approach to (1) enrich the teaching prompt; (2) extend RoboCert to suit the needs of LLM-based model generation, facilitating the definition of properties that reflect common mistakes; (3) and integrating the various tools. Enrichment of the teaching prompt can take advantage of all RoboChart models freely available[7] to cover a wide variety of architectures for the controllers and machines.

Currently, we have not taken advantage of RoboCert to define requirements. Further in the future, we will identify whether use of RoboCert, instead of natural language, can indeed reduce the effort of generation of RoboChart models. Finally, we can provide similar support to automate the evaluation of the RoboCert requirements. Since it has a formal semantics, we can consider the use of animators and other validation approaches to support this work.

Although the work presented here is closely related to the RoboStar suite of notations and frameworks, there is a degree of generality that may be applied across various domains. The core focus of our research pertains to robotics, and this method of generating code for different domain-specific languages could potentially be beneficial for modelling using any requirements language that has a formal semantics and verification support.

Although our preliminary evaluation results do not fully cover the proposed framework, it is a valuable step to demonstrate that LLMs are capable of generating useful RoboChart models. This initial work highlights the potential for refining the process to establish the workflow we propose, to incorporate RoboCert as a readable, but formal, notation, complementing the verification effort.

Related work. The research in [10] has a goal similar to ours. The authors present a proof of concept in which creation of models of multi-agent systems is based on mission specifications and follows an iterative process using an LLM. In contrast, our vision is for models created from requirements using two LLMs and formal verification. The work of [7] examines code generation using LLMs, providing a high-level overview of ongoing developments. The work in [1] investigates

[7] `https://robostar.cs.york.ac.uk/case_studies/`

the generation of DSL code and explores optimisation techniques by assessing whether Retrieval-Augmented Generation can enhance the outcomes produced by LLMs. Our work can complement that effort as well.

References

1. Bassamzadeh, N., Methani, C.: A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation. arXiv preprint arXiv:2407.02742 (2024)
2. Bjerknes, J.D., Winfield, A.F.T.: On Fault Tolerance and Scalability of Swarm Robotic Systems. In: Martinoli, A. (ed.) Distributed Autonomous Robotic Systems - The 10th International Symposium, DARS 2010, Lausanne, Switzerland, November 1-3, 2010. Springer Tracts in Advanced Robotics, pp. 431–444. Springer (2010)
3. Cavalcanti, A., Barnett, W., Baxter, J., *et al.*: RoboStar Technology: A Roboticist's Toolbox for Combined Proof, Simulation, and Testing. In: Software Engineering for Robotics. Ed. by A. Cavalcanti *et al.*, pp. 249–293 (2021)
4. Chkirbene, Z. *et al.*: Large Language Models (LLM) in Industry: A Survey of Applications, Challenges, and Trends. In: IEEE International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET), pp. 229–234 (2024)
5. Gibson-Robinson, T. *et al.*: FDR3 — A Modern Refinement Checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) TACAS. LNCS, vol. 8413, pp. 187–201. Springer, Heidelberg (2014)
6. J. A. Hilder, *et al.*: Chemical Detection Using the Receptor Density Algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **42**(6), 1730–1741 (2012)
7. Joel, S., Wu, J., Fard, F.: A survey on llm-based code generation for low-resource and domain-specific programming languages. ACM Transactions on Software Engineering and Methodology (2024)
8. M. Windsor, A. L. C. Cavalcanti, RoboCert: Property Specification in Robotics. In: A. Riesco, M. Zhang, (eds.) International Conference on Formal Engineering Methods. LNCS, vol. 13478. Springer, Heidelberg (2022). `papers/WC22.pdf`
9. Miyazawa, A., Ribeiro, P., Li, W., *et al.*: RoboChart: Modelling and Verification of the Functional Behaviour of Robotic Applications. Software and Systems Modeling **18**(5), 3097–3149 (2019)
10. Tagliaferro, A., Lestingi, L., Rossi, M.: Preliminary Study of DSL Code Generation for Robotics with LLMs. In: Annual Conference Towards Autonomous Robotic Systems, pp. 273–280 (2025)
11. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle Framework. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) Theorem Proving in Higher Order Logics. LNCS, pp. 33–38. Springer, Heidelberg (2008)
12. Windsor, M., Cavalcanti, A.: RoboCert: Property specification in robotics. In: International Conference on Formal Engineering Methods, pp. 386–403 (2022)
13. Ye, K. *et al.*: Probabilistic modelling and verification using RoboChart and PRISM. Software and Systems Modeling **21**(2), 667–716 (2022)