AI-guided Model-Driven Embedded Software Engineering

Padma Iyenghar[1,2], Friedrich Otte[1] and Elke Pulvermueller[1]
[1]*Software Engineering Research Group, University of Osnabrueck, Germany*
[2]*innotec GmbH, Erlenweg 12, 49324 Melle, Germany*

Keywords: Artificial Intelligence, Model-Driven Engineering (MDE), Embedded Software Engineering (ESE), Unified Modeling Language (UML), Software Development, MDE Tool.

Abstract: In this paper, an use case of Artificial Intelligence (AI) empowered Model Driven Engineering (MDE) in the field of Embedded Software Engineering (ESE) is introduced. In this context, we propose to qualify MDE tools for ESE with an AI assistant or a chatbot. The requirements for the first version of such an assistant and the design concepts involved are discussed. A prototype of such an assistant developed using an open source conversational AI framework used in tandem with an MDE tool for ESE is presented. Empowering MDE tools with such AI assistants, would aid novices in MDE or even non-programmer to learn and adopt model-driven ESE with a not-so-steep learning curve.

1 MOTIVATION

Artificial Intelligence (AI) is a sub-discipline of computer science which is used to supplement technical systems with the ability to process tasks independently and efficiently. With the help of learning algorithms, AI systems can continue learning during ongoing operations, through which the trained models are optimized and the data- and knowledge-bases extended. Recent studies on modelling the impact of AI on the world economy in (Bughin et al., 2018) claim that AI has large potential to contribute to global economic activity. For instance, simulation studies in (Bughin et al., 2018) show that around 70 percent of companies may adopt at least one type of AI technology by 2030 and AI could potentially deliver additional economic output of around $13 trillion by 2030. AI is also starting to impact all aspects of the system and software development lifecycle, from their upfront specification to their design, testing, deployment and maintenance, with the main goal of helping engineers produce systems and software faster and with better quality while being able to handle ever more complex systems. Thus, AI is envisaged to help deal with the increasing complexity of systems and software.

In the direction of the aforesaid context, the Model Driven Engineering (MDE) paradigm has been introduced in the recent decade with the goal of easing the developmental complexity of software and systems. In MDE, models are set in the center of every engineering process. Its target is to guarantee significant rise in productivity, maintenance and interoperability. It is increasingly used in industry sectors such as Cyber Physical Systems (CPS), automotive and aviation to name a few. Thus, MDE has been a means to tame, until now, a part of the aforementioned complexity of software and systems. However, its adoption by industry still relies on their capacity to manage the underlying methodological changes, and also the adoption and training with new tools with significant cost and time overhead. In the recently concluded workshop on AI and MDE (MDE_Intelligence, 2021), it is identified that there is a clear need for AI-empowered MDE, which will push the limits of *classic* MDE and provide the right techniques to develop the next generation of highly complex model-based system.

1.1 Collaboration of AI and MDE

The convergence of two separate fields in computer science such as MDE and AI can give rise to collaboration in two main ways such as (a) AI-guided MDE and (b) MDE for AI. In the following, let us briefly touch upon the opportunities and challenges derived by integration of AI and MDE for both (a) and (b).

For *(a) AI-guided MDE*, MDE can benefit from integrating AI concepts and ideas to increase its power, flexibility, user experience and quality. Some opportunities in this direction can be:

Iyenghar, P., Otte, F. and Pulvermueller, E.
AI-guided Model-Driven Embedded Software Engineering.
DOI: 10.5220/0011006200003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 395-404
ISBN: 978-989-758-550-0; ISSN: 2184-4348

- AI planning applied to (meta-) modeling
- Using machine learning of models, meta-models and model transformation through search-based approaches
- Self-adapting code generators
- AI-based assistants such as bots or conversational agents and virtual assistants for MDE tool. AI-assistant for human-in-the loop modeling, e.g. dialog-based optimization and support for modeling tasks, answering FAQs and tutorials using text-to-text, voice-to-text processing, etc.
- Natural language processing applied to modelling
- Semantic reasoning platforms over domain-specific models
- AI techniques for data, process and model mining and categorization

On the other hand, some challenges for (a) could be in choice, evaluation and adaptation of AI techniques to MDE, such that they provide a compelling improvement to current system, software modeling and generation processes. AI-powered MDE should significantly increase the benefits and reduce the costs of adopting MDE. Furthermore, this step should also enable ease-of-use of MDE tools such that, for instance, they become analogous to use and popularity of low-code platforms in the IT sector.

In the case of *(b) using MDE for AI*, AI can primarily benefit from MDE by integrating concepts and ideas from MDE such as

- Model-driven processes for AI systems development
- Automatic code generation for AI libraries
- Model-based testing of AI artifacts
- Domain-specific modeling approaches for machine learning

Rather than significant challenges, it is expected that experts in MDE can make comprehensive inroads in the AI domain with their rich background and experience in applying MDE across various sectors.

1.2 MDE in ESE

Software development for embedded systems typically involves coding in *programming language C (or C++)*, for a specific *microcontroller*. However, in the recent decade to cope with the growing complexity of software-intensive embedded systems, MDE has become an essential part for analysis, design, implementation and testing of these systems. In the state-of-the-art, a large variety of software modeling practices are used in the domain of Embedded Software Engineering (ESE). A majority of them employ Unified Modeling Language (UML) (OMG, 2021) as a first choice of a graphical formal modeling language. A survey in (Akdur et al., 2018), claims that the top motivations for adopting MDE in ESE (e.g. for CPS development) are cost savings, achieving shorter development time, reusability and quality improvement.

Several state-of-the-practice UML-based MDE tools are available in the ESE domain. While most of these tools claim to help build models quickly, edit programs graphically, generate source code automatically and design systems across platform, they are not necessarily intuitive to immediately put to use in real-life projects after installing them (Sundharam et al., 2021). This makes the learning curve steep and may introduce high cost and time overhead. A typical use case of AI-guided MDE can be foreseen for the aforesaid scenario. For instance, to gain higher acceptance of such MDE tools and bring them a step closer to, for example, a typical embedded software developer venturing to MDE or even to a non-programmer/beginner to learn model-driven ESE, AI-based assistants can be developed and employed together with these tools.

1.3 Novelties

Addressing the aforesaid aspect, we present the following novelties in this short paper:

- Introduce the concept of an AI-assistant or a chatbot for an MDE tool in the context of ESE.
- Define requirements for a first version of such a chatbot and elaborate on the design concepts of one requirement (step-by-step tutorial).
- Present a prototype of the AI assistant developed using RASA (Rasa: Open Source Conversational AI, 2021) for a state-of-the-practice MDE tool in ESE namely, SiSy (Simple System, 2021).

Following this introduction, related work is presented in Section 2. The requirements and design concepts are discussed in section 3. A prototype is presented in section 4 and section 5 concludes this paper.

2 RELATED WORK

2.1 AI-based Assistants

A chatbot is an AI-based program or an assistant, designed to simulate conversation with human users. It uses Natural Language Processing (NLP) to communicate in human language by text or oral speech with humans or other chatbots. Chatbots offer users

comfortable and efficient assistance when communicating with them; they provide them with more engaging answers, directly responding to their problems (Adamopoulou and Moussiades, 2020). A literature review in (Adamopoulou and Moussiades, 2020) presents the history, technology and applications of natural dialog systems or the so-called *chatbots*.

There are two main approaches in developing a chatbot, depending on the algorithms and the techniques adopted, namely, pattern matching and machine learning approaches (Adamopoulou and Moussiades, 2020), (Ramesh et al., 2017).

A chatbot can be developed using programming languages like Java and Python or a chatbot development platform that may be commercial or open-source (Nayyar,D.A., 2019). Open-source platforms make their code available, and the developer can have full control of the implementation. Although, commercial platforms do not give full control to developers, they may still benefit from data for efficient training of the chatbots.

Open source platforms include Rasa (Rasa: Open Source Conversational AI, 2021), Botkit (Botkit: Building blocks for building bots, 2021), Chatterbot (Chatterbot python library, 2021), Pandorabots (Pandorabots: Chatbots with character, 2021) and Botlytics (Botlytics:Analytics for your bot, 2021). Commercial platforms include Botsify (Botsify - A Fully Managed Chatbot Platform To Build AI-Chatbot, 2021), Chatfuel (Chatfuel Chatbot solution, 2021) and Manychat (Manychat:Chat Marketing Made Easy with ManyChat, 2021).

Designing highly functional NLUs requires expert knowledge in machine learning and natural language processing. For this reason, several vendors exist for NLU solutions that make it easier for developers to create programs with NLU. The currently most used and evaluated NLUs are (Abdellatif et al., 2021): *Dialogflow* from Google (Dialogflow, 2021), *LUIS* from Amazon (LUIS-Language Understanding, 2021), *Watson* from IBM (Watson Assistant, 2021) and *Rasa* which is open source (Rasa: Open Source Conversational AI, 2021). These do not only consist of NLUs but also of components for building dialog managers, which makes them full-fledged chatbot frameworks. But until now, there are only scientific evaluations for the NLU part, because it is of greater importance and not every application that needs an NLU also needs a dialog manager.

The results from several evaluation studies show that the performance of the NLU varies greatly depending on the content domain (Canonico and Russis, 2018), (Angara, 2018), (Shawar and Atwell, 2007). Therefore, it is important to determine the suitability of an NLU at the relevant domain. Rasa was specifically evaluated in the context of software engineering by using technical questions asked on stack overflow (Abdellatif et al., 2021). The performance of the NLUs varied from one aspect to another. There was no NLU that outperformed the others on every aspect. In an overall ranking, Watson was placed first, Rasa second, Dialogflow third and LUIS fourth. Watson performed best in intent classification and entity extraction, while Rasa performed best in confidence score. This means that an intent with high confidence value was more often correct for Rasa than it is with other NLUs. This makes Rasa very robust for different confidence thresholds and allows for effective fallback routines. Among the several open source conversational AI framework, we found Rasa to be comprehensive and easy-to-use and supported by its elaborate documentation. Hence, the choice to use Rasa conversational AI framework to build the AI assistant (hereafter referred to as chatbot) for AI-guided MDE in our work was made. An introduction to Rasa framework is not provided here due to space constraints. Interested readers are referred to (Rasa: Open Source Conversational AI, 2021) (Rasa architecture, 2021).

2.2 MDE in ESE: State-of-the-Practice

In the last decade, Model-Driven Architecture (MDA) introduced by the Object Management Group (OMG) (OMG, 2021) is considered as the next paradigm shift in software and systems engineering. Model-driven approaches aim to shift development focus from programming language codes to models expressed in proper domain-specific modelling languages. Thus, models can be understood, automatically manipulated by automated processes, or transformed into other artifacts. For instance, in the direction of adoption of a model-driven approach and the use of simulation-based techniques, significant effort has been spent in the last decade for easing the development and the simulation of complex systems using UML/SysML models in works such as (Bocciarelli et al., 2013), (Bocciarelli et al., 2019), (Sporer, 2015), (Mhenni et al., 2018), (Mhenni et al., 2014) and (Andrianarison and Piques, 2010) to mention a few. Some of these works are also joint efforts from industry and academia.

However, one must admit that the shift from model-based (models used as mere diagrams) to a completely model-driven methodology (models used as central artifacts) in real-life projects in the industry has not yet taken place, especially for ESE domain. For example, in a latest survey in (van der Sanden et al., 2021), a position paper on model-driven

Systems Performance Engineering (SysPE) for Cyber Physical systems (CPS) is presented. The paper concludes that the state-of-practice is model-based and a transition to model-driven SysPE is needed to cope with the ever increasing complexity of today's CPS.

Some state-of-the-practice UML-based MDE tools in the embedded software domain include proprietary tools such as Rhapsody Developer (IBM Software, 2021), Enterprise Architect (Enterprise Architect tool, 2021) and SiSy (Simple System, 2021) and a free UML tool, Visual Paradigm (Visual Paradigm, 2021). In the non-UML domain, Matlab/Simulink (Mathworks Products, 2021) is among the most popular MDE tool employed in the embedded software domain. While most of these tools claim to help build models quickly, edit programs graphically, generate source code automatically and design systems across platform, they are not necessarily intuitive to immediately put to use in real-life projects after installing them.

For the aforementioned scenario, a typical use case of AI-guided MDE can be foreseen. For instance, to gain higher acceptance of such MDE tools and bring them a step closer to, for example, a typical embedded software developer venturing to MDE or even to a non-programmer/beginner to learn model-driven ESE, AI-based assistants can be developed and employed together with these tools.

3 AI ASSISTANT FOR MDE TOOL

As mentioned in section 1.1, in the context of AI guided MDE, the latter can benefit from integrating AI concepts to increase its power, flexibility, user experience and quality. For instance, a starting point can be development of an AI-assistant such as a chatbot, which can serve as conversational virtual assistants for answering FAQs, step-by-step guidance of the tutorials available in the MDE tool, modeling tasks and so on. Such a use case of AI-guided MDE, would help in enabling ease-of-use of MDE tools (e.g. lessen the steep learning curve) and perhaps also help to reduce the costs of adopting MDE.

3.1 Requirements

In the following, requirements for the envisaged AI-assistant for an MDE tool applied in the context of ESE are outlined. However, the overall concepts and ideas discussed here could be applied in the context of an AI-based assistant for *any* MDE tool. Please note that, within the scope of this paper, we make use of an MDE tool SiSy (Simple System, 2021), aiming specifically at MDE for embedded software systems and envisage the usage of the AI-assistant with this tool.

3.1.1 R1: Step-by-Step Guided Tutorial

To enable ease-of-use of the MDE tool step-by-step guidance of tutorials available in the MDE tool can be offered by the AI-based assistant.

- *R1.1-Piece-wise Tutorial:* The tutorial should be provided in a piece-wise manner with step-by step interactive instructions for the user, based on their comfort level.
- *R1.2-Questions at Any Time:* During such a conversation-based tutorial, the AI-assistant should be able to answer questions anytime.
- *R1.3-Manipulate Tutorial State:* The user should be able to request any tutorial, and should also be able to stop or restart the running tutorial or switch to another tutorial. Restarting the tutorial can be useful if the user accidentally gave false information or skipped a tutorial step.
- *R1.4-Context Specific Tutorial:* The chatbot should provide content depending on the context.

3.1.2 R2: Frequently Asked Questions (FAQs)

The chatbot should be able to answer a list of FAQs. These are typically single-turn interactions, which means that the user asks a question and the chatbot can answer in one turn, without additional context information or asking further questions.

3.1.3 R3: Ease of Use

The chatbot must be easy and intuitive to use. It must be clear for the user how to request tutorials and FAQs. This also implies a high robustness for language understanding.

3.1.4 R4: Scalability

The chatbot is envisaged to be used in the long run of the MDE tool. Hence, it is important that it can be easily adapted, i.e., change and add or remove content in a simple and fast way.

3.1.5 R5: Integration

The AI Assistant should be easily accessible while using the MDE tool in question. This can be achieved, for instance, by integrating it within the MDE tool or by making it accessible via the web.

3.1.6 R6: Continuous Improvement

When deployed, the chatbot should continue to collect data and improve its language understanding and dialog management.

3.2 Design Challenges and Decision

This section discusses the design challenges for requirement 3.1.1 only (due to space constraints), to arrive at a design solution. A chatbot usually comprises two main components, namely the NLU and dialog manager. In line with this idea, the proposed design and architecture of the chatbot introduced in this paper is shown in Figure 1.

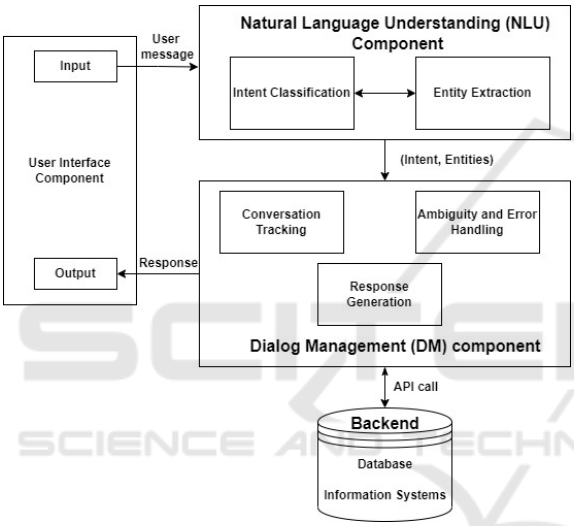

Figure 1: Proposed design employing message handling process of the RASA framework.

The NLU is responsible for understanding the unstructured user input (text information) and the dialog manager controls state and flow of the conversation. As seen in Figure 1, the NLU component takes care of intent classification and entity extraction. The dialog management component handles conversation tracking, ambiguity and error handling and response generation. The user interface component receives the user input, and it is communicated to the NLU unit. Based on the extracted intents and entities, the dialog management component provides a response to the user interface component.

3.2.1 Handling Step-by-Step Tutorial

To provide the tutorial step-by-step as discussed in section 3.1.1, the conversations have to be defined over multiple turns. In contrast, it would be straightforward to provide the conversation in one block, because this can be a simple question/response pattern. The challenge of providing a step-by-step tutorial is increased by the sub-requirement R.1.2 in section 3.1.1, which requires that the questions from the end-user of the AI assistant should be answered any time. This prevents a rigid sequential process where after step one, step two and then step three will follow and so on. This implies a dialog management (cf. Figure 1), which is designed for multiple-turn dialogues, needs to be used. Furthermore, it should be possible to use slots to save information over multiple turns. Rasa's dialog management is designed for multiple-turn dialogues in so far that former intents can be taken into account when the next action is decided. Furthermore, it is possible to use slots to save information over multiple turns in Rasa.

With enough training data, the dialog manager could learn to provide the tutorial steps in the correct order and flexibly react to other questions at the same time. But this approach would be very labor-intensive and there would be no guarantee for success. Hence, other approaches had to be designed: the first approach was to use forms to manage these multiple-turn dialogues, the second approach was to use slots and custom actions. Both these approaches have been designed and evaluated as part of the concept study phase and elaborated below.

Option 1: Form Approach. One approach would be to use forms, since they already are a specific implementation of multiple turn dialogues. Forms are a special type of action that allows to define slots that need to be filled. A form can thus be active over several utterances. As long as there is an empty slot within the form, the agent will ask a predefined question for that slot until it will be filled and move to the next slot. If the user gives an utterance that can not be used to fill the slot, the message will be handled like it would if there was no active form by the dialog manager. This means that the user can ask the same questions he or she normally could, as long as the message is not confused with valid slot input. Therefore, requirement R1.2 would be fulfilled.

Forms are usually used to collect user information in a structured fashion. After the bot gives a response, it will turn back to the form and repeat the question for the currently empty slot. Figure 2 shows how a form could be used to implement step-by-step tutorials by using slots in a slightly different way. The form starts after the NLU component detects the intent to start a tutorial. Then, the first step of the tutorial will be provided, and the user will be asked to fill the first slot with something like *"Did you succeed?"* or *"Do you want to continue to the next step?"*. Only if the next

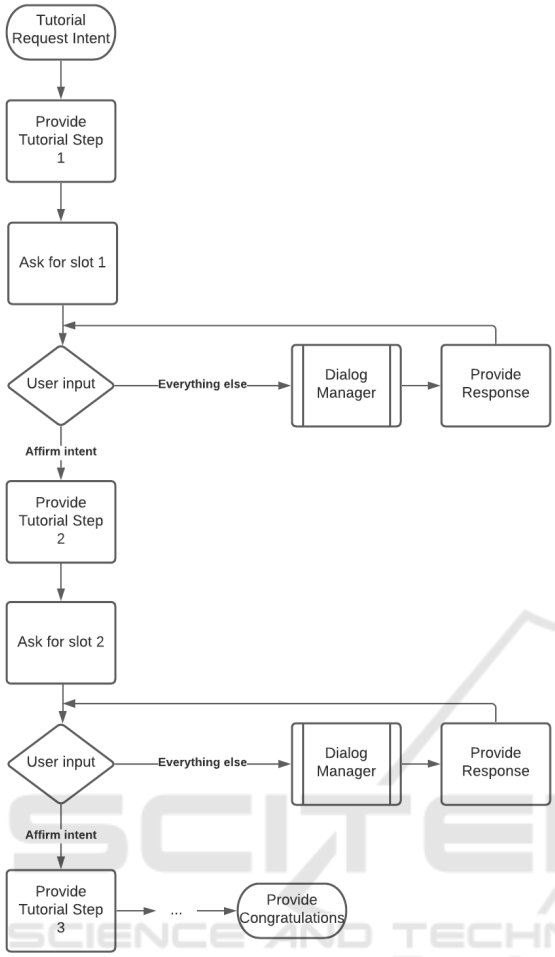

Figure 2: Flowchart diagram of the form approach option to implement step-by-step tutorials.

user input is interpreted as affirmative intent by giving a message like *"yes"* or *"please continue"* will the form continue to the next step. If no affirmative intent is detected, the message will be handled by the dialog manager. This is repeated until all tutorial steps have been provided. If necessary, the first slots can be used to collect context information like controller type to fulfill requirement R1.4.

A prototype evaluation of this approach has shown that this approach in fact fulfills requirements R1.1 and R1.2 and R3, but also has shown some disadvantages. Some disadvantages are, for each tutorial, a form has to be created and for each tutorial step, an individual slot has to be created. Furthermore, for each form, a *FormValidationAction* event that includes the validation logic for each slot, has to be defined. This makes adding content very time consuming and is in conflict with requirement R4. The effort could probably be mitigated by a script that adds all these slots and the validation logic automatically, but this would have to be updated every time that Rasa changes the domain syntax or structure or the form structure.

Option 2: Custom Actions Approach. Another solution could be to use custom actions to provide the next tutorial step and slots to save the current state of the tutorial. The tutorial state could be modeled by the current tutorial name and the current tutorial step. When a tutorial is requested, a tutorial specific custom action will be executed. This will allow handling tutorial specific business logic, like different content depending on the controller type. A specially created *"next"* intent will be used to trigger a custom action *"Tutorial dispatcher"* that evaluates the current tutorial slot and calls the respective custom action. This approach is shown in Figure 3.

As in option 1, the process begins after NLU module detects the intent to start a tutorial. The custom action will increment the tutorial step counter slot and set the slot that defines the current tutorial. If context information is needed to provide the tutorial, a form is run and all the necessary information will be collected. Otherwise, the agent will provide the first step of the tutorial directly. When the user says something like *"next"*, the *"Tutorial Dispatcher"* will be called.

The purpose of this action is to request the "current tutorial" slot and dispatch to the tutorial specific action. It could also be used to handle tutorial switches and edge cases. Calling the tutorial specific action will increment the tutorial step counter again and provide the next step. If the user has a question, the bot will answer it and the slots will remain untouched. The *tutorial dispatcher* decides which custom action to start, depending on the current tutorial slot. If a tutorial needs to ask user information, forms can be used. They can be invoked by the custom action before the first step, so the collected information can be used in the subsequent steps.

Decision. As seen above, two approaches are evaluated for the requirement R1 in section 3.1.1. Although the form approach utilizes an existing Rasa tool, it comes with great disadvantages. The process of adding new tutorials would be very compartmentalized, and there is no obvious way to remedy this. The second approach using custom actions has a similar problem, because for each tutorial, a custom action has to be added. But the implementation effort can be reduced by making use of object inheritance.

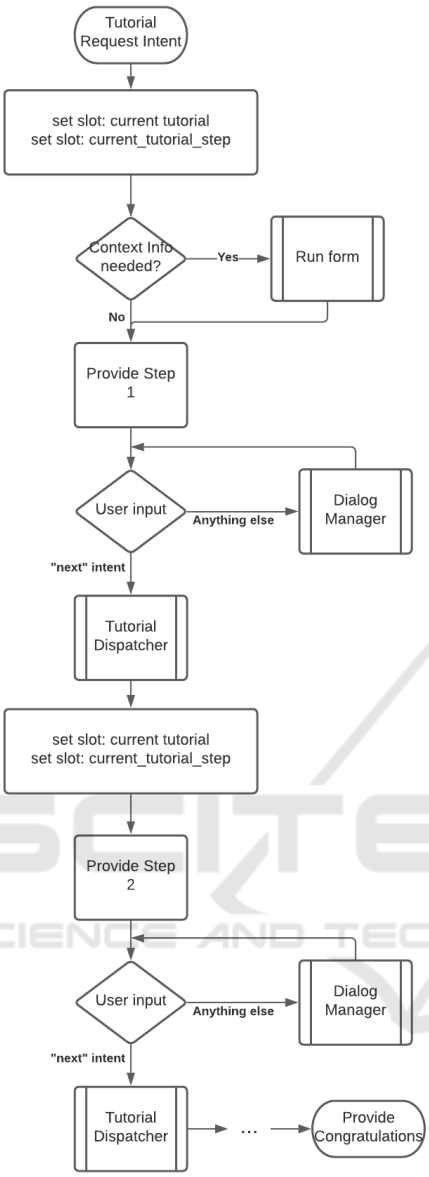

Figure 3: Flowchart diagram of the custom action approach option to implement step-by-step tutorials.

4 PROTOTYPE

The design alternatives and corresponding design decisions for the requirements mentioned in section 3.1.1-3.1.6 have been implemented in the chatbot prototype. A first version of the prototype of the chatbot can assist the MDE tool (SiSy) user with a set of tutorials and answer a set of FAQs, as seen in Figure 4.

Due to space limitations, only the implementation specifics of the requirement R1 in section 3.1.1 is presented in this section.

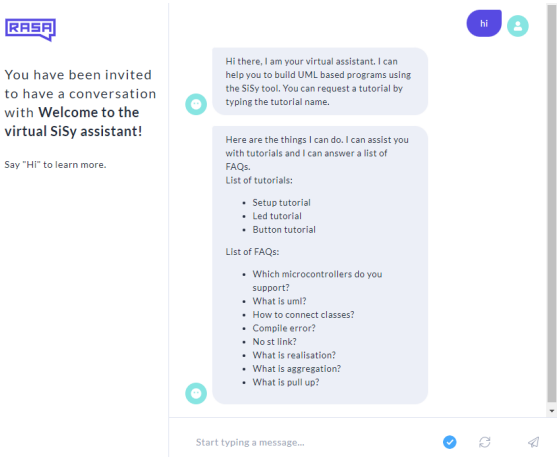

Figure 4: Welcome screen of the chatbot with a list of its capabilities.

4.1 Step-by-Step Tutorial

The SiSy tool provides various tutorials to learn Model-driven implementation of embedded software. In the first version of the prototype, three tutorials are provided (Figure 4). In this section, the design and implementation of a message handling mechanism where a multi-step tutorial can be followed, and the chatbot is able to answer questions at any time is presented.

Figure 5 shows the design of basic structure of tutorial implementation using the LED tutorial example (i.e., toggling LEDs in embedded target) example and described below. Further, the series of steps in the step-by-step LED tutorial provided piece-wise by the chatbot for the MDE tool SiSy (as a conversation-based tutorial) is shown in Figure 6 and Figure 7.

- A *Tutorialhhandler* class has been implemented to reduce implementation effort involved in adding new tutorials (i.e., for scalability, extensibility and continuous improvement). Adding a tutorial always requires a new custom action. These actions are very similar to attributes and functions.
- To add a new tutorial, one has to create a new *Subclass* to the *TutorialHandler* class and overwrite the name function and *init* function. Note that, the tutorials are split into multiple text messages and presented to the user based on the user input during conversation with the chatbot.
- For instance, for the *LED Tutorial,* a custom action *handle_led_tutorial* has been created (see actions.py in Figure 5). The function of this custom action is to provide the next step of the LED tutorial. The action gets the current step of the LED tutorial by querying the tracker for a specific slot.

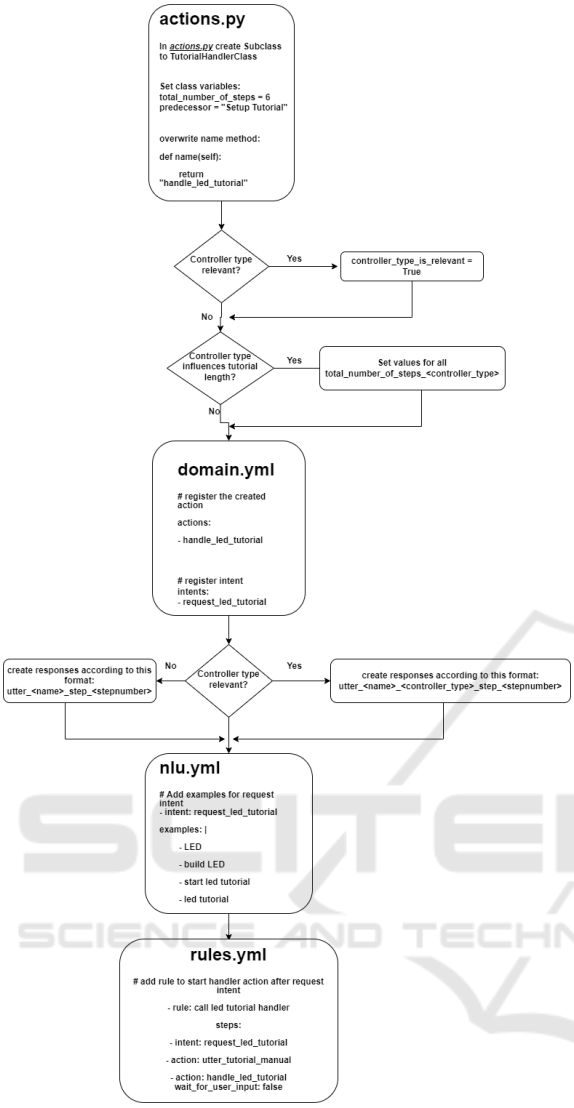

Figure 5: Design of basic structure of tutorial handling in chatbot (here example is LED Tutorial).

If the action was called for the first time, it should first start a form.

- The form defines all information that is needed by the user in order to provide the correct tutorial steps. Here the user is asked about the controller type he has. Based on the user input, (and if the current step is greater than zero and smaller than the maximum number of steps), the next tutorial step will be uttered. The name of the utterance for each step follow the pattern *utter tutorial name step number*.
- Based on the extracted information from user input, the form could be used here to provide fine-grain instructions tailored to the user information. If the total number of steps is reached for a tutorial, a congratulation message will be uttered.
- In addition, two rules have been added for the dialog manager (see rules.yml in Figure 5). The first rule: there is a 'request led form intent', the LED form handler is called. This is how the tutorial will be started. The second rule: Whenever there is a 'next' intent, the 'dispatch tutorials' action will be called. This will check which tutorial is active and dispatch to the correct action.

This architecture described above is flexible since it allows the user to switch between multiple tutorials. Whenever the user wants to change the current tutorial, the user can type the name of the tutorial he wishes, and it will start.

5 CONCLUSION

In this paper, an use case of AI-driven MDE is presented. A proposal to qualify MDE tools for ESE with an AI assistant or a chatbot is outlined. The requirements of a conversational chatbot which uses text (for input and output) and image (only output) as the conversational medium was elaborated. The design alternatives and design choices made to fulfil one of the requirements (step-by-step tutorial) is presented in this short paper. In the prototype, the design of a basic structure for message handling mechanisms in the chatbot is described. It was showcased how a multi-step tutorial can be followed by the user and how the chatbot is able to answer questions at any step.

This is only the tip of the iceberg. The bot can be further improved in umpteen ways. One possible use case is by making use of training data for the dialog management, with the so-called *stories*. The bot can help the user at a specific tutorial step, even if the user does not ask for specific information. For example an *'unspecific problem'* intent could be added with example messages like *'It does not work', 'nothing works', 'how does this work?', 'I have a problem'* and so on. In the next step, the stories could be revised and the responses that the chatbot should give in the exact situation - depending on the current tutorial, the current step or even slot values - are added to the stories. After enough examples, the dialog manager learns to provide the necessary information even if the user does not ask for it specifically-thereby achieving a true AI guided model-driven development experience.

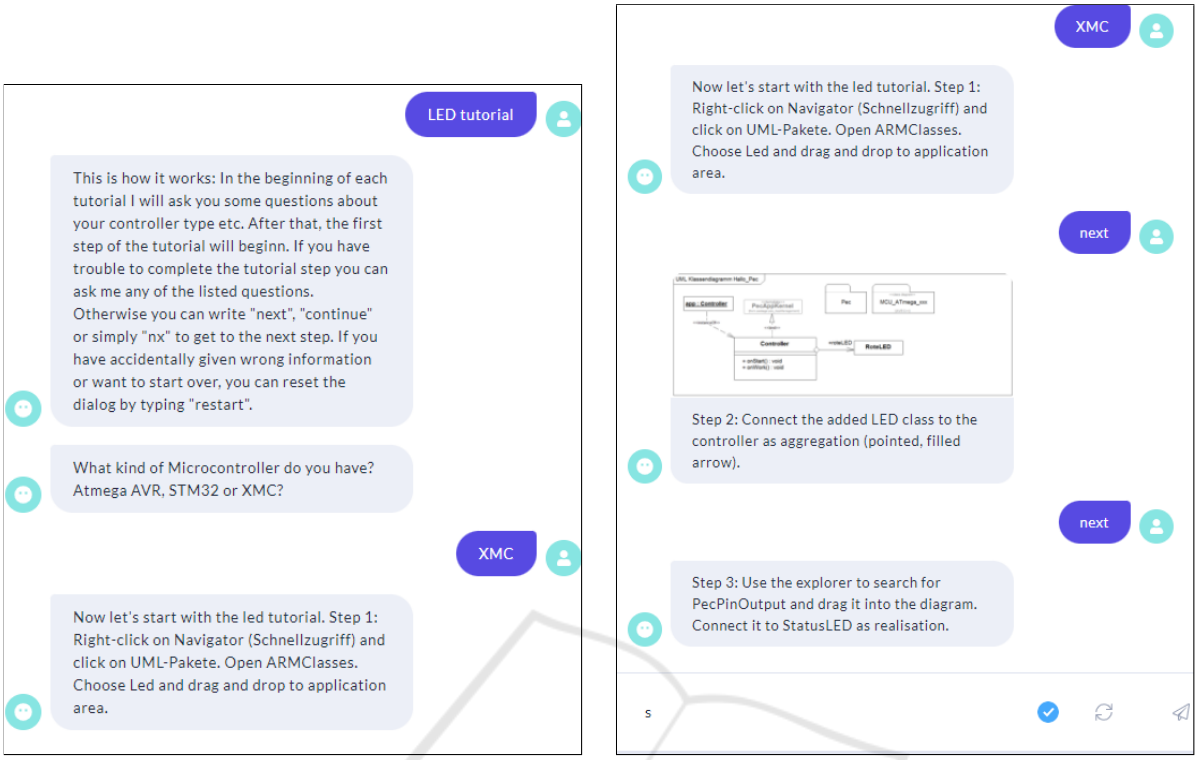

Figure 6: Step (a)-left and step (b)-right in the step-by-step LED tutorial provided piece-wise by the chatbot for the MDE tool SiSy.

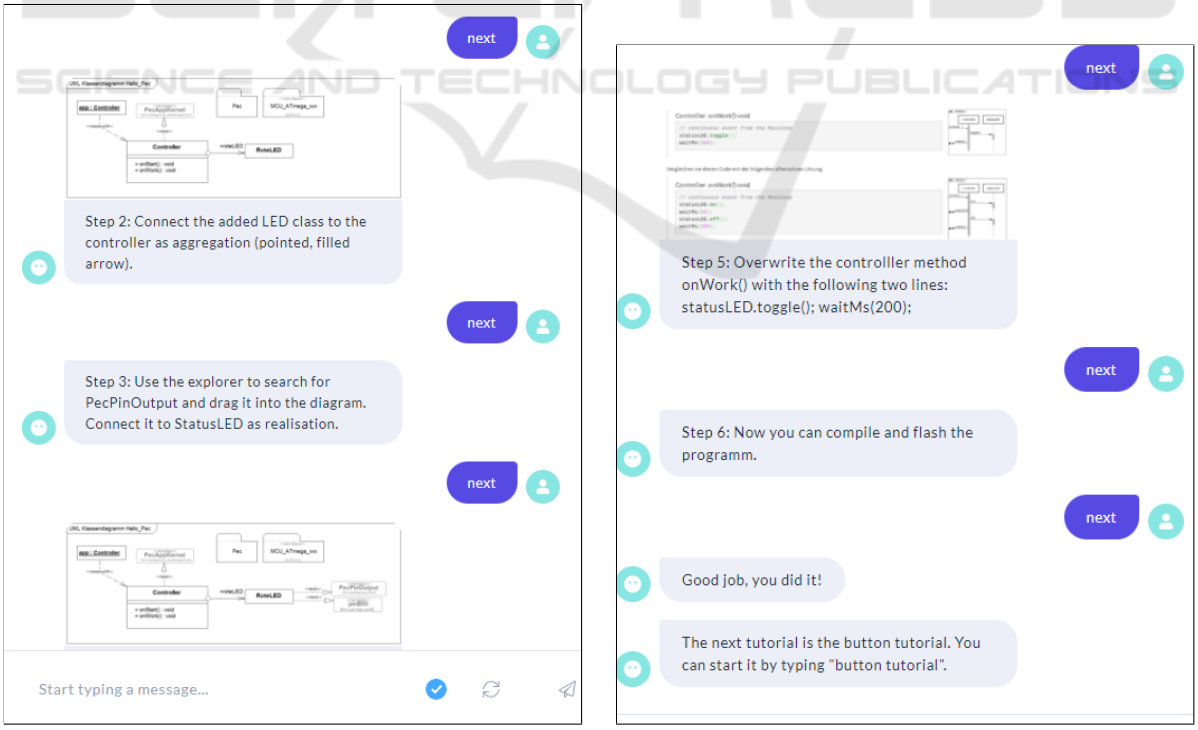

Figure 7: Step (c)-left and step (d)-right in the step-by-step LED tutorial provided piece-wise by the chatbot for SiSy tool.

REFERENCES

Abdellatif, A., Badran, K., Costa, D., and Shihab, E. (2021). A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering (early access)*.

Adamopoulou, E. and Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006.

Akdur, D., Garousi, V., and Demirörs, O. (2018). A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture*, 91:62–82.

Andrianarison, E. and Piques, J.-D. (2010). Sysml for embedded automotive systems : a practical approach.

Angara, P. P. (2018). Towards a deeper understanding of current conversational frameworks through the design and development of a cognitive agent.

Bocciarelli, P., D'Ambrogio, A., Giglio, A., and Gianni, D. (2013). A saas-based automated framework to build and execute distributed simulations from sysml models. In *2013 Winter Simulations Conference (WSC)*, pages 1371–1382.

Bocciarelli, P., D'Ambrogio, A., Giglio, A., and Paglia, E. (2019). Model-driven distributed simulation engineering. In *Proceedings of the Winter Simulation Conference*, WSC '19, page 75–89. IEEE Press.

Botkit: Building blocks for building bots (Last accessed: 21.11.2021). https://botkit.ai/.

Botlytics:Analytics for your bot (Last accessed: 21.11.2021). https://www.botlytics.co/.

Botsify - A Fully Managed Chatbot Platform To Build AI-Chatbot (Last accessed: 21.11.2021). https://botsify.com/.

Bughin, J., Seong, J., Manyika, J., Chui, M., and Joshi, R. (2018). Notes from the AI frontier: Modeling the impact of AI on the world economy. https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-modeling-the-impact-of-ai-on-the-world-economy. [Online; accessed 17-Nov-2021].

Canonico, M. and Russis, L. D. (2018). A comparison and critique of natural language understanding tools.

Chatfuel Chatbot solution (Last accessed: 21.11.2021). https://chatfuel.com/ .

Chatterbot python library (Last accessed: 21.11.2021). https://chatterbot.readthedocs.io/en/stable//.

Dialogflow (Last accessed: 21.11.2021). https://cloud.google.com/dialogflow/.

Enterprise Architect tool (2021). http://www.sparxsystems.com/. Accessed 17-Nov-21.

IBM Software (2021). Ibm rational rhapsody developer. https://www.ibm.com/software/products/en/ratirhap. Accessed 17-Nov-21.

LUIS-Language Understanding (Last accessed: 21.11.2021). https://docs.microsoft.com/en-us/azure/cognitive-services/luis/.

Manychat:Chat Marketing Made Easy with ManyChat (Last accessed: 21.11.2021). https://manychat.com.

Mathworks Products (Last accessed: 21.11.2021). https://www.mathworks.com/.

MDE_Intelligence (2021). 3rd Workshop on Artificial Intelligence and Model-driven Engineering. https://mde-intelligence.github.io/. [Online; accessed 17-Nov-2021].

Mhenni, F., Choley, J.-Y., Penas, O., Plateaux, R., and Hammadi, M. (2014). A sysml-based methodology for mechatronic systems architectural design. *Advanced Engineering Informatics*, 28(3):218–231. Multiview Modeling for Mechatronic Design.

Mhenni, F., Nguyen, N., and Choley, J.-Y. (2018). Safesyse: A safety analysis integration in systems engineering approach. *IEEE Systems Journal*, 12(1):161–172.

Nayyar,D.A. (2019). Chatbots and the Open Source Tools You Can Use to Develop Them. https://www.opensourceforu.com/2019/01/chatbots-and-the-open-source-tools-you-can-use-to-develop-them/.

OMG (Last accessed: 21.11.2021). Object management group. https://www.omg.org/.

Pandorabots: Chatbots with character (Last accessed: 21.11.2021). https://www.pandorabots.com/.

Ramesh, K., Ravishankaran, S., Joshi, A., and Chandrasekaran, K. (2017). A survey of design techniques for conversational agents. In Kaushik, S., Gupta, D., Kharb, L., and Chahal, D., editors, *Information, Communication and Computing Technology*, pages 336–350, Singapore. Springer Singapore.

Rasa architecture (Last accessed: 21.11.2021). https://rasa.com/docs/rasa/arch-overview/.

Rasa: Open Source Conversational AI (Last accessed: 21.11.2021). https://rasa.com/.

Shawar, B. A. and Atwell, E. (2007). Different measurement metrics to evaluate a chatbot system. In *HLT-NAACL 2007*.

Simple System (2021). https://sisy.de/. Accessed 17-Nov-21.

Sporer, H. (2015). A model-based domain-specific language approach for the automotive e/e-system design. RACS, page 357–362, New York, NY, USA. Association for Computing Machinery.

Sundharam, S. M., Iyenghar, P., and Pulvermueller, E. (2021). Software architecture modeling of autosar-based multi-core mixed-critical electric powertrain controller. *Modelling*, 2(4):706–727.

van der Sanden, B., Li, Y., van den Aker, J., Akesson, B., Bijlsma, T., Hendriks, M., Triantafyllidis, K., Verriet, J., Voeten, J., and Basten, T. (2021). Model-driven system-performance engineering for cyber-physical systems. In *Proceedings of the 2021 International Conference on Embedded Software*, EMSOFT '21, page 11–22, New York, NY, USA. Association for Computing Machinery.

Visual Paradigm (2021). https://www.visual-paradigm.com/solution/freeumltool/. Accessed 17-Nov-21.

Watson Assistant (Last accessed: 21.11.2021). https://www.ibm.com/uk-en/products/watson-assistant.