Towards Low-code Architecture and Development of Embedded Systems

Felipe Xavier
Dept. of Math. and Computer Science
Eindhoven University of Technology
Eindhoven, Netherlands
f.d.azeredo.coutinho.xavier@tue.nl

Loek Cleophas
Dept. of Math. and Computer Science
Eindhoven University of Technology
Eindhoven, Netherlands
l.g.w.a.cleophas@tue.nl

Michel R. V. Chaudron
Dept. of Math. and Computer Science
Eindhoven University of Technology
Eindhoven, Netherlands
m.r.v.chaudron@tue.nl

***Abstract*—While the interest in low-code platforms grows, these technologies have mostly focused on administrative business applications, and other domains remain relatively untouched. We think much can be gained from introducing low-code concepts and tools to the embedded systems domain, which still suffers from a relatively complex development cycle. This paper presents a roadmap for creating an approach that allows the generalization of embedded software development through a low-code platform. We highlight five main tasks required to accomplish this endeavor, which will be researched in the near future: 1) studying existing low-code development characteristics; 2) defining a reference architecture for embedded systems; 3) identifying prevailing embedded system design patterns; 4) designing a multi-view approach with a consistent visual language; and 5) integrating formal verification into the platform.**

***Index Terms*—low-code, embedded systems, architecture**

I. Introduction

Recently, the "low-code" software development approach has gathered considerable attention. This approach simplifies software creation by leveraging graphical modeling languages to define applications. It then employs code generators to build these applications, thus requiring minimal coding expertise from users [1]. Although Forrester Research introduced the term "Low-Code" in 2014 [2], it made its debut in the scientific literature in 2017 and has attracted growing interest within the research community following Zolotas et al. [3] in 2018.

While the interest in this technology grows, it has focused on administrative business applications, and other domains remain essentially untouched [4]. Much can be gained from introducing low-code concepts and tools to the embedded systems domain, which still suffers from a relatively complex development cycle [5]. Given the importance of these systems, which are present in key areas such as healthcare and automotive, a technology that simplifies their development and guarantees their correctness is crucial.

Some approaches attempt to make inroads in applying low-code to embedded domains; Tools such as MetaEdit+ [6] and MontiCore [7] already provide a good level of support to develop software for embedded systems; however, these solutions are dependent on language designers to tailor such a workbench for each specific application. A generic low-code solution inspired by what Mendix[1] and OutSystems[2] offer for administrative business applications could greatly improve and facilitate the development of embedded systems.

To create such a generic solution, it is necessary to first define what the key concerns are of an embedded system. For example, for administrative business applications, the Mendix platform breaks down its software architecture into three key layers: user interface layer, business logic layer, and data layer, and provides three different Domain-Specific Languages (DSLs) with each of them used to define one such layer. In the background, Mendix facilitates the information exchange while developing these "layers" so that developers feel like they are using a single language. This standardized architecture is what allows for the seamless integration of the different layers. However, while the 3-layered reference architecture is well-defined for business applications, that is hardly the case for embedded systems. Thus, before creating a low-code development platform for embedded systems, a generic architecture must be defined.

A. Is low-code the right approach?

A restrictive architecture seems to go against common practices in the embedded domain, which is more diverse than business applications or web development [8], [9]. Nonetheless, low-code presents great benefits to the embedded domain.

First, embedded systems are generally considered to have a higher entry barrier than other software domains. The restrictive hardware concerns of embedded systems are nearly nonexistent in, e.g. web or application development. With an increasing demand for embedded developers due to emerging Internet of Things (IoT) and "smart" consumer products [10], [11], a low-code approach to embedded development could lower the required skill level and attract more developers.

Furthermore, low-code provides an easy and accessible way of reusing software artifacts. Developers often create their own repositories for commonly used components in embedded systems, for example, an I^2C communication bus class or a thread pool. While useful, this keeps knowledge limited to a small number of developers. A low-code platform abstracts away such low-level concepts, which are usually reused, providing a more intuitive way of maintaining them.

Finally, restricting the possible implementations of these systems makes it easier to verify their correctness. There

[1]https://www.mendix.com
[2]https://www.outsystems.com

is no shortage of techniques to formally verify embedded systems, be it through timing measurements [12] or model checking [13]; however, they still suffer from state-explosion problems mainly due to the sheer number of possible actions in a system [14]. Thus, a low-code approach not only facilitates software development through reusability, it has the potential to facilitate the development of formally correct software by limiting the size of the state space of the systems.

B. Research questions and paper structure

In this context, this paper presents a roadmap for creating a low-code approach that allows the generalization of embedded software development through a low-code platform. To achieve the proposed goal, we consider as main research question:

RQ How to design a low-code approach for the development and generation of embedded software?

This question leads us to five sub-questions:

RQ1 What are the main characteristics of low-code development platforms, including their implementation, and how are these characteristics applicable to embedded software development?

RQ2 What does a generic reference architecture for embedded software systems look like?

RQ3 What patterns can be used to generalize and abstract embedded software into concepts necessary and suitable for a low-code development platform?

RQ4 What are the requirements for designing user-friendly visual languages (low-code) for embedded systems?

RQ5 How can formal analysis be integrated into a low-code development platform to allow the verification of embedded systems?

The remainder of this paper is structured as follows: Section II presents related works and discusses their differences with and similarities to this paper. Section III describes the proposed vision and delineates the necessary steps. Finally, Section IV provides the concluding remarks.

II. Related work

This section presents the current state of the art for the different subtopics related to this research: DSLs and Model-Based Design (MBD) for embedded systems, reference architectures for such systems, and, finally, their verification.

In the context of DSLs for embedded systems, Hammond et al. [15] present *HUME*, a DSL for real-time embedded systems. This DSL focuses on low-level system requirements and tackles concepts essential to the domain, such as concurrency, low resources, and determinancy. Mizzi et al. [16] focus on distributed embedded systems. The authors propose *D'Artagnan*, a DSL that attempts to abstract away low-level implementation and allow the compilation, analysis, transformation, and interpretation of high-level descriptions of real-time stream processing applications. Both of these examples highlight essential concepts in developing embedded systems; however, their textual approach still limits understandability compared to graphical modeling languages [17], [18].

We find various other DSL approaches for embedded systems, but most focus on more specific sub-domains of embedded systems. Related to the field of robotics, we encounter work by Wesselink et al. [19] and Trezzy et al. [20] who create DSLs to simplify the creation of Robot Operating System (ROS) components by abstracting the standardized steps of ROS development. Still, in the robotics field, Heinzemann et al. [21] presents a DSL for the definition of formally verifiable robot tasks. Related to IoT, we find the methodology described by Ihirwe [4], which attempts to encompass the entire development cycle of IoT applications in low-code. Finally, Panayiotou et al. [22] introduces *SmAuto*, a DSL for creating and executing IoT applications. The authors use a layered approach to abstract away the implementation details and heterogeneity inherent to the IoT domain.

Concerning understandability and MBD, many authors have applied MBD to embedded systems. Similar to the aforementioned DSL works, these papers attempt to bring MBD to the field of embedded systems. Pietrusewicz et al. [23] presents a meta-modeling approach for cyber-physical systems through the *ResearchML* language, which attempts to improve traceability in development projects. Related to this are the already mentioned MetaEdit+ and MontiCore platforms, introduced by Kelly et al. [6] and Krahn et al. [7] respectively. While these platforms do not focus directly on embedded systems, they have both been used for such systems, as shown by Fischer et al. [24] and Dalibor et al. [25]. As discussed, although extremely powerful, these tools still require defining a reference architecture for each system and have limited support for reusability. This, in turn, complicates the verification and validation of such systems. Other works explore specific fields in the domain of embedded systems. For example, Dhouib et al. [26] developed *RobotML*, a modeling approach that supports not only the development of robotic systems but also their deployment and testing through simulations.

Regarding reference architectures, Loukil et al. [27] suggest utilizing Architectural Description Language (ADL) to define the architecture for embedded systems. The authors discuss how a higher level of abstraction provides a better overview of scalable and complex embedded systems. In a similar vein, Wang et al. [28] propose a reference architecture based on reusable components defined by finite state machines. The paper describes a system where different components can be easily connected, similar to the FMI standard[3]. Wang et al. [29] follow a similar concept but focus on the simulation of such systems. As with the other subtopics of interest, the literature tends to focus on specific subfields of embedded systems. McDuffie et al. [30], for example, describe architectures of spacecraft attitude control systems. Fischer et al. [24] and Heinecke et al. [31] focus on automotive systems. Finally, Heisey et al. [32] present a reference architecture for drones.

The final subfield of related work we consider is that of verification of embedded systems. The work of Bartocci et al. [12] presents the *MoonLight* tool for monitoring temporal and spatiotemporal properties of mobile and spatially dis-

[3]https://fmi-standard.org

tributed cyber-physical systems. This approach enhances current verification techniques for temporal properties with spatial constraints. Miyazawa et al. [13] introduce *RoboChart* for modeling and verification of the functional behavior of robotic applications. Their research proposes a set of constructs for modeling robotic applications that supports verification via model checking. The approach, however, is once again constrained and not expanded to other fields of embedded systems. Besides these, the aforementioned work of Heinzemann et al. [21] allows the creation of verifiable robot tasks through their DSL, while Shaukat et al. [33] demonstrate a method for the validation of the architecture of robots based on ROS. Finally, Antonio et al. [34] discuss the validity of several verification techniques when applied to code generated for embedded systems through SysML. These works are relevant, as some of the techniques can be generalized to embedded systems and applied to code generated by low-code platforms.

Finally, it is important to highlight Node-RED [35], SmowCode [36], DIME [37], as well as the works of Brouzos et al. [38] and Bourr et al. [39]. Node-RED is a flow-based programming tool conceived for the IoT context that aims to wire and connect hardware devices, APIs, and online services. SmowCode is a commercial tool with a similar goal but focused on the ESP32 microcontroller. DIME is foremost a low-code/MBD platform for the development of web applications; however, it has proven to be useful as well in the context of digital twins [40] and IoT [41]. Brouzos et al. propose a low-code solution for defining components and communication for robots that use the ROS middleware. The authors utilize the aforementioned Node-RED tool to exemplify the communication between the ROS components. Lastly, Bourr et al. integrate X-Klaim [42], a coordination language designed to model and program distributed systems, ROS and BPMN [43], and, thus, provide a method for defining multi-robotic missions through a visual language.

III. A LOW-CODE PLATFORM FOR EMBEDDED SYSTEMS

The overall goal of this research is to create a low-code development platform that enhances the creation of embedded software systems. We devise a plan that we believe will provide the basis to achieve this. In this section, we present and discuss our hypotheses and steps to tackle each research question defined in Section I. Thus, this section goes through 1) defining the main characteristics of low-code development platforms and applying these characteristics to embedded software development; 2) proposing an initial reference architecture and a plan to refine it; 3) identifying embedded system patterns to be reused in a low-code platform; 4) delve into user-friendliness aspects of graphical languages; and, finally, 5) discussing formal verification in low-code platforms.

A. Low-code development characteristics

Before defining a new platform, it is necessary to understand what already exists and which technologies that are used in existing low-code development can be leveraged for embedded systems. As mentioned, the great strength of low-code is having patterns that generalize application development.

To investigate RQ1, we start by looking into the Mendix example already discussed. Similar to other platforms, Mendix provides a clear separation of goals in application development. Figure 1 provides an overview of the 3-layered structure followed by the Mendix platform. A similar separation of concerns can be applied to embedded systems if there is an underlying reference architecture. This is a key concept necessary for successfully developing the desired low-code platform and the focus of the next section.

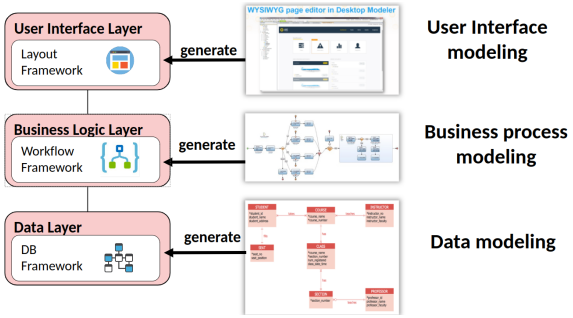

Fig. 1. Recreation of the structure enforced by the Mendix platform

While most low-code technologies focus on business applications, a few projects target different types of systems. For example, the Node-RED tool, discussed in Section II, is used for IoT applications. Among its key characteristics is its web-based visual editor built on Node.js[4], which allows it to be used from any browser and deployed in nearly any hardware. Furthermore, since Node.js is a widely adopted technology, it lowers the barrier to entry and allows for easy extensibility. Besides allowing users to extend the tool with their own nodes, Node-RED provides blocks for the most commonly used patterns in the IoT context, such as performing HTTP requests and MQTT messaging.

Bock et al. [44] discuss how these characteristics identified in Node-RED are shared by most low-code development platforms. Their study also highlights other common characteristics, such as support for defining roles, integration with textual programming for advanced users, and definition of complex data structures. We hypothesize that all of these characteristics can be used in an embedded systems context and should, thus, be part of a low-code platform focused on that domain. As the next step to answering RQ1, a state-of-the-practice study will be performed to identify and translate these characteristics into the embedded systems domain.

B. A reference architecture for embedded systems

As discussed, the reference architecture allows us to define the standardized workflow of the low-code platform. Based on existing works, mostly discussed in Section II, it is possible to start answering RQ2 and stipulate what a reference architecture for embedded systems looks like.

After investigating several reference architectures in the literature, we believe that a combination of the reference

[4]https://nodejs.org/en

architectures of Nakagawa et al. [45] and Eklund et al. [46] provide a good basis for our research. The reference architecture of Eklund et al. focuses on lower-level hardware abstractions, while the work by Nakagawa et al. provides a very generic layered reference for the whole system, which considers the functions of each layer. Furthermore, while many studied architectures shared common characteristics, the one by Nakagawa et al. was the only one that explicitly considered the system evolution, one of our focuses with the low-code platform. An overview of the hypothesized reference architecture is provided in Figure 2.

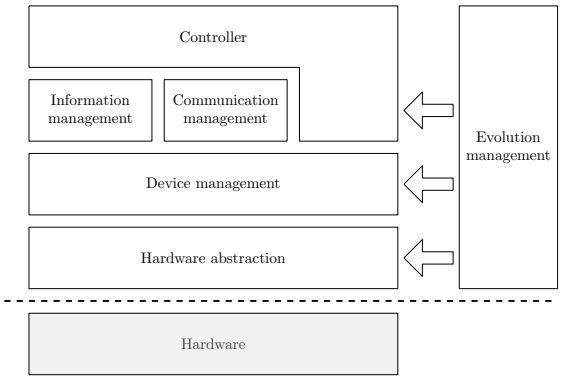

Fig. 2. Layers of the proposed reference architecture combining the works of Nakagawa et al. [45] and Eklund et al. [46]

From the most abstract viewpoint, the reference architecture follows a layered structure. At the lowest level, there is an abstraction of the hardware components themselves, e.g., sensors and actuators, as proposed by Eklund et al. On top of that, the architecture for robots discussed by Nakagawa et al. is followed. Then, a device management layer is used to centralize communication with the hardware components and provide a mechanism for easily introducing or removing them.

Above the device management lay the information and communication management layers. The information management layer handles persistent data, for example, the interface with a database. The communication management layer ensures the exchange of volatile data with other devices or external controllers, for example, when distributed embedded systems must coordinate a shared mission.

Next is the controller layer, which controls the system's behavior. This layer interprets the information retrieved through sensors, combines it with existing knowledge, and takes the necessary actions to complete the received tasks. Radestock and Eisenbach [47] formalized such behavior in the 4Cs (Computation, Communication, Coordination, and Composition), which is widely used in embedded systems. Bruynickx [48] later extended these 4Cs with "Configuration" and described its uses in robotic applications, see Figure 3. Despite its focus on robotics, such uses can be generalized for embedded systems that must also interact with the world and often have some form of perception, planning, monitoring, and control. We can think, for example, about smart devices and IoT.

While these layers are enough to describe many embedded systems, Nakagawa et al. discuss an additional concept of an

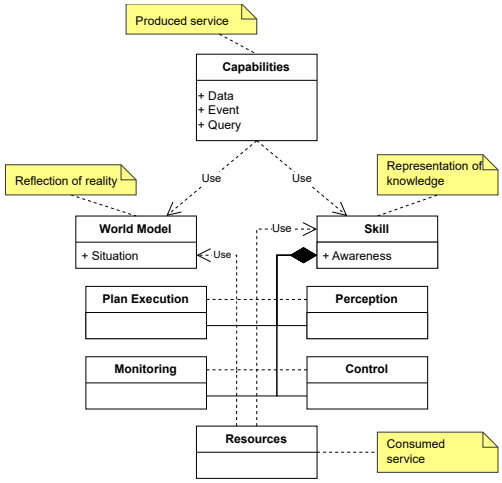

Fig. 3. Class diagram of the task structure proposed by Bruynickx [48]

"evolution manager." This manager is responsible for handling the replacement of components in the system, for example, the substitution of the controller algorithm. Moreover, it handles the adaptation to better execute its current tasks. With the current interest in AI [49] and its capabilities related to continuous improvement, we deemed it important to add this layer to our architecture as it might become more prominent.

This reference architecture is our starting point. As the next step in our endeavor, software architecture recovery and functional decomposition techniques will be applied to extract the underlying software architectures from different open and closed-source software repositories. This aims to create an empirically grounded and realistic reference architecture. Based on the encountered reference architectures, it will be possible to define more grounded requirements and workflows for a generic low-code development platform for embedded systems.

C. Embedded systems patterns

However, before we discuss the requirements and workflows for such a low-code development platform, we must mention patterns often used in embedded systems development that such a platform could leverage. While a reference architecture is essential for a streamlined low-code solution, other generalizable aspects of software development can greatly facilitate the creation of such systems.

These patterns are still unclear, but much work has already been done on communication, long-term storage, parallel task scheduling, and resource allocation. As is the practice of low-code and one of its key strengths, these patterns can be abstracted away from the user. Similar to RQ1, for RQ3, we plan a study into the state of practice to formalize the necessary embedded system patterns required in a low-code platform targeted towards that domain.

D. User-friendly graphical languages

Our research aims to combine the reference architecture and embedded development patterns into a user-friendly platform. Although these patterns are not yet clear and require further

research, it is important to extract their key concepts and what they represent and make them clear to the users.

As discussed, graphical DSLs are an essential part of low-code; thus, our future research will focus on the design of such a language. Based on the theoretical reference architecture, a one-language-per-layer approach is suggested; instead of using a single DSL to represent the system's different concerns, we plan to develop separate languages to simplify the interaction with the devices, internal control, information management, etc.

A low-code approach, due to the aforementioned abstractions, takes away finer control over these patterns. A balance between abstraction and customization is essential to attract both new and experienced users. The end goal of using low-code is so that users do not need to think about these implementation specificities and can focus on the system's goal. That is, the *what* and not the *how*, as implied by the separation of concerns design principle.

As Caire et al. [50] discuss, there are key concerns when developing a visual language. These concerns become more significant when multiple languages must interact with one another. Therefore, we plan to use MetaEdit+ as inspiration and separate the visual aspect from the semantics of the DSLs. With the DSLs defined textually, it is possible to focus on integrating them in a visually cohesive manner. The end goal is to make use of these languages seamlessly, so little mental effort is required to adapt to the development of a new layer.

E. Formally correct low-code

As discussed in Section II, it is easier to formally verify a system with a smaller scope, that is, a system with limited expressiveness and behavior. With the proposed approach, we expect to facilitate the verification of embedded systems.

Related to this easiness, there is one, still speculative, last step in our plan, which is to define a visual DSL for requirement specification. While formal methods are widespread and can be relatively easily generated, system requirements are very specific for a single system. Therefore, users still need a friendly way to define these formal requirements to be analyzed. This question is still open, and how such a language would look has not been explored much; nonetheless, we added it here to provide a clear global picture of our end goal. The mentioned work on BPMN by Bourr et al. [39] provides an idea of what we would like to achieve.

IV. Conclusion

With this paper, we lay down a vision to create a low-code development platform for embedded systems. This technology that attracts ever more attention is still lacking in the domain of embedded systems, so we hope this paper functions as a starting point for future research in the area by pinpointing key concepts necessary for its full realization.

First, existing low-code patterns and technologies must be identified and reused to reduce future work. These include abstracting recurring domain design patterns, ensuring a clear separation of design, modeling, and generation, and facilitating easy extensibility. Concurrently, a reference architecture that encompasses the key aspects of embedded systems must be defined. This should be based on theoretical idealism as well as practical concepts used in the development of such systems.

This reference architecture can finally be represented with separate DSLs for the different concerns. Despite being different, each DSL is intended to share a single workflow that combines visual and textual programming to minimize the platform's learning curve. While there is still a lot to be researched, by following the proposed guidelines, we expect to create a low-code development platform that supports the design, implementation, and deployment of safe and reliable embedded software systems.

Acknowledgment

This research is part of the Cynergy4MIE project and was funded by the Chips Joint Undertaking and its members, including top-up funding provided by National Authorities under Grant Agreement No. 101140226.

References

[1] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 376–381, 2019. 10.1016/j.ifacol.2019.10.060.

[2] C. Richardson, J. Rymer, C. Mines, A. Cullen, and D. Whittaker, "New Development Platforms Emerge For Customer-Facing Applications," Tech. Rep. 15, Forrester, Cambridge, 2014.

[3] C. Zolotas, K. C. Chatzidimitriou, and A. L. Symeonidis, "RESTsec: a low-code platform for generating secure by design enterprise services," *Enterprise Information Systems*, vol. 12, no. 8-9, pp. 1007–1033, 2018. 10.1080/17517575.2018.1462403.

[4] F. Ihirwe, "Low-Code Engineering for the Internet of Things," *SSRN Electronic Journal*, 2023. 10.2139/ssrn.4539001.

[5] A. Al Tareq, M. T. Hossain, Z. Al Dodaev, and A. Haque, "Prospects and Challenges of Embedded Systems for Semiconductor Devices in Industry - A Case Study," *Control Systems and Optimization Letters*, vol. 2, pp. 144–149, June 2024. 10.59247/csol.v2i1.97.

[6] S. Kelly, K. Lyytinen, and M. Rossi, "MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment," in *Advanced Information Systems Engineering*, (Berlin, Heidelberg), pp. 1–21, Springer, 1996. 10.1007/3-540-61292-0_1.

[7] H. Krahn, B. Rumpe, and S. Völkel, "MontiCore: a framework for compositional development of domain specific languages," *Int. Jrnl. on Software Tools for Technology Transfer*, vol. 12, no. 5, pp. 353–372, 2010. 10.1007/s10009-010-0142-1.

[8] P. Koopman, "Embedded system design issues (the rest of the story)," in *Proceedings ICCD*, pp. 310–317, Oct. 1996. 10.1109/ICCD.1996.563572.

[9] M. Smith, J. Miller, L. Huang, and A. Tran, "A More Agile Approach to Embedded System Development," *IEEE Software*, vol. 26, pp. 50–57, May 2009. 10.1109/MS.2009.57.

[10] G. Andersen and MoldStud Research Team, "The rise in demand for embedded software engineers in various industries." https://moldstud.com/articles/rise-in-demand, Jan 2024. Last accessed: December 2024.

[11] Ciklum, "Five challenges of embedded technology systems & why experts are the key to success." https://www.ciklum.com/resources/blog/five-challenges-of-embedded-technology-systems-why-experts-are-the-key-to-success, 2024. Last accessed: December 2024.

[12] E. Bartocci, L. Bortolussi, M. Loreti, L. Nenzi, and S. Silvetti, "MoonLight: A Lightweight Tool for Monitoring Spatio-Temporal Properties," 2021. 10.48550/arXiv.2104.14333.

[13] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, and J. Woodcock, "RoboChart: modelling and verification of the functional behaviour of robotic applications," *Software & Systems Modeling*, vol. 18, no. 5, pp. 3097–3149, 2019. 10.1016/j.pmcj.2024.101931.

[14] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, pp. 429–528, Berlin, Heidelberg: Springer, 1998. 10.1007/3-540-65306-6_21.

[15] K. Hammond and G. Michaelson, "Hume: A Domain-Specific Language for Real-Time Embedded Systems," in *Generative Programming and Component Engineering*, (Berlin, Heidelberg), pp. 37–56, Springer, 2003. 10.1007/978-3-540-39815-8_3.

[16] A. Mizzi, J. Ellul, and G. Pace, "D'Artagnan: An Embedded DSL Framework for Distributed Embedded Systems," in *Proceedings of the RWDSL 2018*, RWDSL2018, (USA), pp. 1–9, ACM, 2018. 10.1145/3183895.3183899.

[17] C. Chen, P. Haduong, K. Brennan, G. Sonnert, and P. Sadler, "The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages," *Computer Science Education*, vol. 29, no. 1, pp. 23–48, 2019. 10.1080/08993408.2018.1547564.

[18] N. Hollmann and S. Hanenberg, "An Empirical Study on the Readability of RegEx: Textual Versus Graphical," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 74–84, 2017. 10.1109/VISSOFT.2017.27.

[19] B. Wesselink, K. de Vos, I. Kuertev, M. Reniers, and E. Torta, "RoboSC: a domain-specific language for supervisory controller synthesis of ROS applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9090–9096, 2023. 10.1109/ICRA48891.2023.10161436.

[20] M. Trezzy, I. Ober, I. Ober, and R. Oliveira, "Leveraging domain specific modeling to increase accessibility of robot programming," in *2021 IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM)*, pp. 1–9, 2021. 10.1109/ECMSM51310.2021.9468864.

[21] C. Heinzemann and R. Lange, "vTSL - A Formally Verifiable DSL for Specifying Robot Tasks," in *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (Madrid), pp. 8308–8314, IEEE, 2018. 10.1109/IROS.2018.8593559.

[22] K. Panayiotou, C. Doumanidis, E. Tsardoulias, and A. L. Symeonidis, "SmAuto: A domain-specific-language for application development in smart environments," *Pervasive and Mobile Computing*, vol. 101, p. 101931, 2024. 10.1016/j.pmcj.2024.101931.

[23] K. Pietrusewicz, "Metamodelling for Design of Mechatronic and Cyber-Physical Systems," *Applied Sciences*, vol. 9, no. 3, p. 376, 2019. 10.3390/app9030376.

[24] A. Fischer, J.-P. Tolvanen, and R. T. Kolagari, "Automotive Cybersecurity Engineering with Modeling Support," in *19th Conf. on Computer Science and Intelligence Systems (FedCSIS)*, pp. 319–329, 2024. 10.15439/2024F5017.

[25] M. Dalibor, M. Heithoff, J. Michael, L. Netz, J. Pfeiffer, B. Rumpe, S. Varga, and A. Wortmann, "Generating customized low-code development platforms for digital twins," *Journal of Computer Languages*, vol. 70, p. 101117, 2022. 10.1016/j.cola.2022.101117.

[26] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "RobotML, a DSL to Design, Simulate and Deploy Robotic Applications," in *Simulation, Modeling, and Programming for Autonomous Robots*, (Berlin), pp. 149–160, Springer, 2012. 10.1007/978-3-642-34327-8_16.

[27] S. Loukil, S. Kallel, B. Zalila, and M. Jmaiel, "Toward an Aspect Oriented ADL for Embedded Systems," in *Software Architecture*, pp. 489–492, Springer, 2010. 10.1007/978-3-642-15114-9_47.

[28] S. Wang and K. G. Shin, "An architecture for embedded software integration using reusable components," in *Proceedings of the 2000 CASES*, CASES '00, (USA), pp. 110–118, ACM, 2000. 10.1145/354880.354896.

[29] Y. Wang, X. Zhou, Y. Dong, and C. Li, "A MDA-Based Approach for General Embedded Software Simulation Platform," in *2009 SCALCOM-EMBEDDEDCOM*, pp. 20–25, 2009. 10.1109/EmbeddedComScalCom.2009.14.

[30] J. McDuffie, "Using the architecture description language MetaH for designing and prototyping an embedded spacecraft attitude control system," in *20th DASC. (Cat. No.01CH37219)*, vol. 2, pp. 8E3/1–8E3/9 vol.2, 2001. 10.1109/DASC.2001.964241.

[31] H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Maté, K. Nishikawa, and T. Scharnhorst, "AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures," *Int. Congress & Exposition On Transportation Electronics*, 2004.

[32] C. W. Heisey, A. G. Hendrickson, B. J. Chludzinski, R. E. Cole, M. Ford, L. Herbek, M. Ljungberg, Z. Magdum, D. Marquis, A. Mezhirov, J. L. Pennell, T. A. Roe, and A. J. Weinert, "A Reference Software Architecture to Support Unmanned Aircraft Integration in the National Airspace System," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1, pp. 41–55, 2013. 10.1007/s10846-012-9691-8.

[33] N. Shaukat, S. Dubey, B. Kaddouh, A. Blight, L. Mudrich, P. Ribeiro, H. Araujo, R. Richardson, L. Dennis, A. Cavalcanti, and M. Mousavi, "Trustworthy ROS Software Architecture for Autonomous Drones Missions: From RoboChart Modelling to ROS Implementation," in *2024 20th IEEE/ASME Int. Conf. on Mechatronic and Embedded Systems and Applications (MESA)*, pp. 1–7, 2024. 10.1109/MESA61532.2024.10704818.

[34] E. A. Antonio, R. Rovina, and S. C. P. F. Fabbri, "Verification and Validation Activities for Embedded Systems - A Feasibility Study on a Reading Technique for SysML Models:," in *Proceedings of the 16th International Conference on Enterprise Information Systems*, (Lisbon, Portugal), pp. 233–240, SCITEPRESS - Science and and Technology Publications, 2014. 10.5220/0004887302330240.

[35] O. Foundation and Contributors, "Node-red: Low-code programming for event-driven applications." https://nodered.org/, 2019. Last accessed: December 2024.

[36] A. Mewada, "Smowcode." https://smowcode.com, 2021. Last accessed: January 2025.

[37] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, and B. Steffen, "DIME: A Programming-Less Modeling Environment for Web Applications," in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, (Cham), pp. 809–832, Springer, 2016. 10.1007/978-3-319-47169-3_60.

[38] R. Brouzos, K. Panayiotou, E. Tsardoulias, and A. Symeonidis, "A Low-Code Approach for Connected Robots," *Jrnl. of Intelligent & Robotic Systems*, vol. 108, no. 2, p. 28, 2023. 10.1007/s10846-023-01861-y.

[39] K. Bourr, F. Tiezzi, and L. Bettini, "Model-Driven Development of Multi-Robot Systems: From BPMN Models to X-Klaim Code," in *Leveraging Applications of Formal Methods, Verification and Validation. Rigorous Engineering of Collective Adaptive Systems*, (Cham), pp. 224–242, Springer, 2024. 10.1007/978-3-031-75107-3_14.

[40] T. Margaria and A. Schieweck, "The Digital Thread in Industry 4.0," in *Integrated Formal Methods*, (Cham), pp. 3–24, Springer International Publishing, 2019. 10.1007/978-3-030-34968-4_1.

[41] H. A. A. Chaudhary, I. Guevara, J. John, A. Singh, T. Margaria, and D. Pesch, "Low-Code IoT Application Development for Edge Analytics," in *IoT through a Multi-disciplinary Perspective*, (Cham), pp. 293–312, Springer International Publishing, 2022. 10.1007/978-3-031-18872-5_17.

[42] L. Bettini, E. Merelli, and F. Tiezzi, "X-Klaim Is Back," in *Models, Languages, and Tools for Concurrent and Distributed Programming: Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pp. 115–135, Cham: Springer International Publishing, 2019. 10.1007/978-3-030-21485-2_8.

[43] OMG, "Business Process Model and Notation (BPMN), Version 2.0," tech. rep., Object Management Group, Joulukuuta, 2011.

[44] A. C. Bock and U. Frank, "In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms," in *2021 ACM/IEEE MODELS-C)*, pp. 57–66, 2021. 10.1109/MODELS-C53483.2021.00016.

[45] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, "Consolidating a Process for the Design, Representation, and Evaluation of Reference Architectures," in *2014 IEEE/IFIP ICSA*, (Australia), pp. 143–152, IEEE, 2014. 10.1109/WICSA.2014.25.

[46] U. Eklund and J. Bosch, "Architecture for embedded open software ecosystems," *Jrnl of Systems and Software*, vol. 92, pp. 128–142, 2014. 10.1016/j.jss.2014.01.009.

[47] M. Radestock and S. Eisenbach, "Coordination in evolving systems," in *Trends in Distributed Systems CORBA and Beyond*, (Berlin, Heidelberg), pp. 162–176, Springer, 1996. 10.1007/3-540-61842-2_34.

[48] H. Bruyninckx, *Building blocks for complicated and situational aware robotic and cyber-physical systems with common sense*. Leuven University Press, in press. To be published.

[49] L. Floridi, "Why the AI Hype is Another Tech Bubble," *Philosophy & Technology*, vol. 37, no. 4, p. 128, 2024. 10.1007/s13347-024-00817-w.

[50] P. Caire, N. Genon, P. Heymans, and D. L. Moody, "Visual notation design 2.0: Towards user comprehensible requirements engineering notations," in *2013 21st IEEE RE)*, pp. 115–124, jul 2013. 10.1109/RE.2013.6636711.