Towards Generating Model-Driven Speech Interfaces for Digital Twins

Ramya Jayaraman, Daniel Lehner, Stefan Klikovits, Manuel Wimmer
Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT)
Institute of Business Informatics – Software Engineering
Johannes Kepler University Linz, Austria
{firstname.lastname}@jku.at

***Abstract*—The recent decade saw enormous advances of the capabilities of speech synthesis and speech recognition. While specific benefits depend on the individual applications, speech interfaces typically increase accessibility, enable "hands-free" and "no-screen" interaction, and often increase interaction speed and allow for more flexible usage patterns and increased multitasking, leading to higher user satisfaction.**

This paper presents a method to transfer these powerful benefits to the digital twin (DT) domain by automatically generating speech interfaces for DT applications. Our approach is based on Model Driven Engineering principles, where we automatically deduce command patterns from structural model information as used in, e.g., DT platform specifications. The speech interface generation is highly configurable and extendable, thus it can be applied to different DT models. The concepts behind the generator are generic as well, thus they can be ported to other applications and platforms. We validate our approach by applying it to two DT demonstration cases and provide a detailed description of the sketch interface configuration workflow.

***Index Terms*—speech recognition, speech interfaces, model-driven engineering, interface generation, digital twin interface**

I. Introduction

Digital twins (DTs) [1] are virtual replicas of (typically) cyber-physical systems (CPSs). In their most basic form, DTs can be likened to Internet of Things (IoT) applications, where a computer system provides a virtual counterpart to an *actual system's* (a.k.a. the physical twin's) sensors and actuators, thus displaying system measurements and offering interaction capabilities. Nonetheless, modern DTs surpass these elementary capabilities and allow for much more advanced workflows such as simulation, monitoring, system calibration and optimisation, data analysis, and machine learning.

To aid the creation and management of DT systems, a handful of so-called DT platforms recently emerged (e.g., Azure Digital Twins, AWS IoT TwinMaker and Eclipse Ditto). These platforms provide industry-grade stability, security and scaling [2] and offer structural modelling languages that are reminiscent of UML's class and object diagrams for the systems' configuration [3]. Recent research is working towards bringing Model Driven Engineering (MDE) [4] research benefits to these DT systems.

Nonetheless, a significant challenge of any DT application remains the creation of an adequate and consumer-friendly user interface (UI). CPSs nowadays reach never-before-seen levels in terms of number of devices, complexity, and geographical and logical size, while edge devices become ever-more intertwined with our daily lives, requiring new forms of user interaction.

The recent decade provides major advances of speech synthesis and speech recognition [5]. While the specific advantages depend on the individual applications, speech interfaces typically increase accessibility, enable "hands-free" and "no-screen" interaction, often increase interaction speed and while allowing for more flexible usage patterns and increased multitasking. It is thus natural, that these advantages should also be exploited for DT applications.

This paper presents a novel method to bring these powerful benefits to the DT world by automatically generating a speech interface for a DT application. Our approach is based on MDE principles, where we automatically deduce command patterns from structural DT model information. Specifically, we use extended variants of UML class and object diagrams to model the system, from which we generate the speech commands. Users actively receive verbal feedback, providing confirmation of correct command execution (as opposed to, e.g., silent speech interfaces [6]). Our speech interface generation is easily configurable and extendable so that it can be applied to different systems. The concepts behind the generator are generic and can be ported to other applications and platforms.

Our contributions can be summarized as follows: 1. We provide an approach to automatically derive speech interfaces for DT systems from structural system models, namely UML class/object diagrams. 2. We describe three types of information that should be added to these models to increase user-friendliness and integration into legacy systems. Specifically, object, attribute and operation *aliases*, REST *endpoint* definitions and *custom actions*. 3. We apply our approach to two DT case study systems to showcase its feasibility[1].

The rest of this paper is structured as follows: Section II provides information about the scientific background and the DT systems we use as motivation. Section III describes the general approach of our model-driven speech interface framework. Section IV evaluates our approach. Section V compares our framework to related work before Section VI concludes.

[1]See also https://youtu.be/7u45YXOH-jc

II. Background and Motivating Examples

A. Modelling Digital Twins

DTs are virtual replications of physical assets—often referred to as physical twins (PTs)—with a bi-directional communication between the DT and the PT [1]. The DT is usually represented as a model containing information about data structures, geometry or physics of the PT [7]. In this paper, we use structural information of the PT to automatically configure the available speech interactions. We base the generated speech interface on UML class diagrams, as they have already been used in the literature to represent this structural aspect of DTs [8], [3], besides other similar languages such as AutomationML [9], [10].

Current interfaces focus on configurable graphical interfaces (e.g., [11] or [12]) or RESTful APIs that can be called by software programs, which are usually provided by industrial DT platforms [2]. In this paper, we extend this state-of-the-art with the advantages that come with the application of speech interfaces for interacting with DTs.

B. Speech Interfaces

The introduction of machine learning and artificial intelligence in the domain of voice-enabled personal assistant programs such as Apple's Siri, Amazon's Alexa and Microsoft's Cortana in smartphones, automobile entertainment and smart home systems pushed their interaction capabilities to new heights. While before, only a fixed set of strictly matched commands was available, modern systems enable "almost natural" voice-based system interaction.

Their popularity yields a fresh interest in porting these capabilities to other applications and enhancing them with native language support. Frameworks such as React Speech Recognition[2] and Web Speech API[3] provide speech capabilities to virtually any application on the web, and integration platforms such as Make[4] allow to easily connect these speech capabilities with legacy systems.

Nonetheless, such frameworks typically rely on manually crafted speech modules, which are based on careful definitions. Our goal is to avoid this step and reuse existing information from structural system data. This data often already exists in the form of DT system models or can be extracted from REST API definitions or using automated code analysis tools and methods such as [13].

C. Motivating Examples

Our approach is motivated by the following considerations. First, while graphical user interface (GUI) applications are seen as the de-facto standard, they often do not provide the required accessibility for all users. Second, new users may often be overwhelmed with the amount of information shown in typical CPS' GUIs. A speech-based workflow may guide the users more easily and create or increase familiarity. Third, in certain industries or situations where people are required to operate in synergy with robots, it is not always possible to look at a screen or use manual input devices. Thus, a non-visual and hands-free interaction can largely increase the applications' usability.

[2]https://www.npmjs.com/package/react-speech-recognition
[3]https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
[4]https://www.make.com/

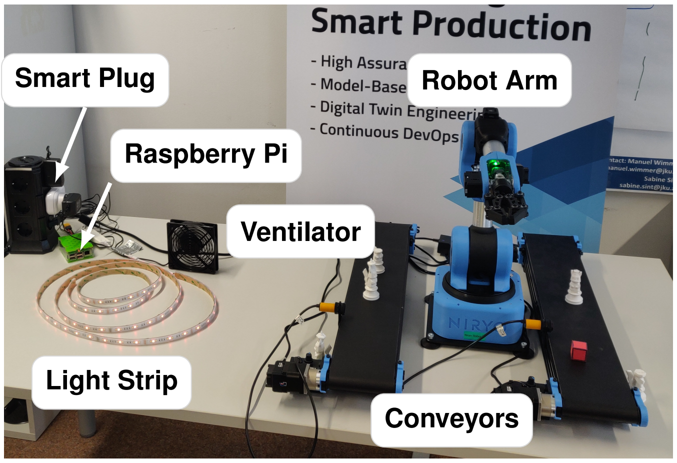


Fig. 1: Left: (simplified) smart room system with ventilator, light strip and Raspberry Pi-based air quality measurements. Right: robot system with 6-DOF robot arm and two conveyor belts.

In the following, we describe two example DT systems, for which we develop speech interfaces, namely a smart room system and a robot application (see Figure 1). Note that both systems existed already before our research on this topic. Hence, the speech interfaces should to be flexible enough to accommodate for legacy design choices.

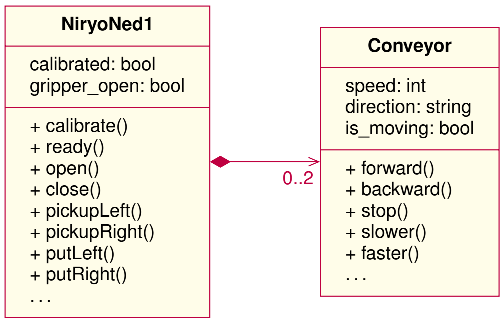
a) Robot system: Our first system is an application that represents an industrial robot system. It consists of a *Niryo Ned*[5] 6 degree-of-freedom (6-DOF) robot arm, equipped with a gripper and two attached conveyor belts. The system's purpose is to transport objects along the conveyor belts and move them from one conveyor to the other using the robot arm.

For simplicity, in our setting, we assume that the conveyor belts always move in parallel, i.e. any REST command always (de-) activates both of them. Figure 2 shows both the UML class (2a) and object diagrams (2b), showing the system's devices' capabilities (operations) and attributes.

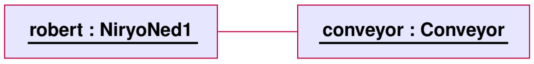
The robot's REST API controls were originally developed in Python using the `FastAPI` and `pyniryo` libraries.

b) Smart room system: In our second use case, the implemented DTs represent a smart room and indoor air quality application [14]. Specifically, we use Raspberry Pis and off-the-shelve IoT technology (smart plugs, Zigbee/MQTT, etc.) for our application. The smart room contains an automated lighting system (represented by an LED strip) and an air quality system (represented by a ventilator). Additionally, it is possible to read the air quality data from the room system.

[5]https://niryo.com/products-cobots/ned-six-axis-robot-arm/



(a) Class diagram (reduced)



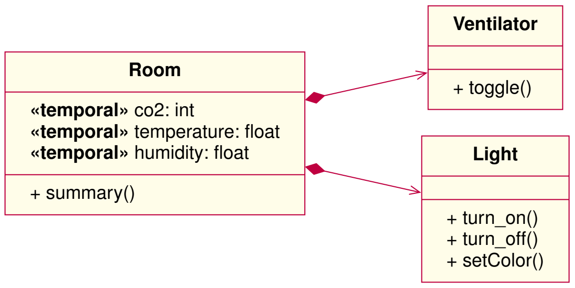
(b) Object diagram

Fig. 2: Robot system – UML class diagram and object diagram

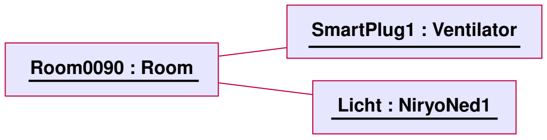
This system is also controlled via a REST API but additionally is connected to a database which allows the polling of historic data (e.g., yesterday's temperature readings).

Figure 3 displays the class and object diagrams of the smart room system. Note the **«temporal»** stereotype (inspired by [15]) which is used to represent attributes where historical data is stored.

The use cases are implemented and published to the open source GitHub repository, detailed descriptions for setup and usage of the whole infrastructure can be referred from online repository[6].



(a) Class diagram (reduced)



(b) Object diagram

Fig. 3: Smart room system – UML class diagram and object diagram

[6]https://github.com/derlehner/IndoorAirQuality_DigitalTwin_Exemplar

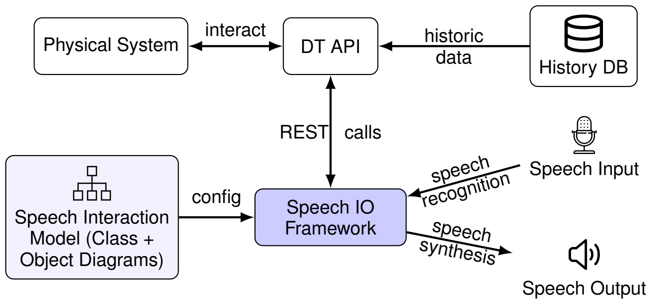


Fig. 4: Speech Interface Workflow for DTs

III. Speech-based DT Interaction Framework

A. Architecture

Figure 4 provides a general overview of the speech-based DT interaction framework proposed in this paper. On the upper half, we see a simplified DT application, consisting of PT, DT REST API and a database that stores historical values. Note that the figure omits all DT components that are not used for the speech interface (e.g., controllers, DT services, etc). The bottom half visualises the core of our approach, namely the *Speech IO Framework*, which is configured from (extended) class and object diagrams. This interface offers both speech recognition (speech-to-text) and speech synthesis (text-to-speech) functionality to enable interactions with the available PT. The speech framework UI is connected to the PT via the RESTful web service that offers (i) execution of operations on the PT, (ii) access to current values for each attribute of the PT, and (iii) access to aggregated historical data for **«temporal»** attributes. Information on the available PTs, operations and attributes of these PTs, as well as the mapping of this information to the specific REST endpoints that have to be called to get this data, are all extracted from the *speech interaction model*, i.e., the class and object diagrams.

B. The Speech Interaction Model

The basic interaction functionality is directly extracted from the UML class and object diagrams. For simplicity, in this publication, we presume that each diagram is available as one individual file. Furthermore, our model is assumed to be in a highly simplified JSON format, as exemplified in Listing 1[7]. Note that we expect that in actual production systems, the information will be extracted from standardized formats such as EMF's XML/XMI [16] or JSON artefacts [17].

Using the information from the diagrams, the app automatically configures the speech interface to listen to the following commands:

Device list: The app can produce a list of all known devices (taken from the object diagram).

Device type: We treat the UML classes as device types. Mentioning a device type yields a list of devices of this type, as defined in the object diagram.

[7]The full models are available on Github: https://github.com/cdl-mint/SpeechInterface_DigitalTwins/tree/main/src/UML_Models

Device name: When a device name (i.e., an element of the object diagram) is mentioned without any further information, the app responds with the available operations and attributes.
Device operations: A device name + an operation name results in calling that operation.
Device attributes: Device name + an attribute will fetch the attribute's current value.
Attribute aggregation: By mentioning a device, an attribute and an aggregator (e.g., mean, minimum, maximum) the historic data is aggregated. This can be further customized by mentioning certain time spans (e.g., today, yesterday, this/last week).

All the information for these interactions can be automatically extracted from the system models, rendering our approach flexible and easily maintainable.

a) Extensions: Additionally to the extraction of e.g. device and operation names, we discovered that the speech interface requires additional information. This enables both the configurability for legacy systems, but also increases user-friendliness. In particular, we added the following three types of information to the UML diagrams.

Extension 1: Class, attribute & operation name aliases. We note that next to the default name (e.g. an ID or function name), in many cases is is convenient to use "friendly names" for devices and operations (e.g. we might prefer to say "living room lamp" instead of "SmartLamp003"). Furthermore, it can become tedious to remember exact operation and attribute names. *Aliases* may be easily added to offer alternative names and thereby increase usability. For instance, we may "flick", "switch" or "toggle" our ventilator, and it will every time trigger the same action. Similarly, "velocity" and "speed" will poll the same attribute of the conveyors, and a robot arm should perform the same action, independent of whether we tell it to "put down" or "place" the object it is holding. It is important, though, that aliases should be unique to a class.
Extension 2: Mapping to API endpoints. As the speech interface is being developed for legacy applications, we note that these systems contain "legacy design choices". Hence, the speech interaction models might not exactly match the existing REST API endpoints. The models can thus be extended by providing specific `endpoint` information for each attribute and operation, thereby allowing users to maintain compatibility with these systems. Similarly, a `data` field can be used to add necessary REST payload data.
Extension 3: Custom actions. In some situations, users may choose to implement customized actions for specific calls. For instance, the light strip in our smart room system can change the colour of the light. To enable this functionality, we thus provided a custom function that converts common colour names to the RGB format that is required by the API.

Note that Extensions 2 and 3 are of particular interest for systems where the REST API cannot be easily changed. These include legacy systems or systems that are not under control of the speech interface developer.

Listing 1: Speech interaction model – class diagram (excerpt)

```
[{
  "name": "Conveyor",
  "extends": "Device",
  "attributes": [{
    "name": "speed",
    "temporal": false,
    "aliases": ["velocity"],
    "type": "int",
    "endpoint": "/Robots/Conveyor",
    "action": "RobotAttribute"
  },
  ... -- more attributes
  ],
  "operations": [{
    "name": "run",
    "aliases": ["move", "on", "start"],
    "action": "RobotAction",
    "endpoint": "/Robots/Conveyor/",
    "data": { "forward": true, "on": true
        }
  }, {
    "name": "moveback",
    "aliases": ["back", "backward"],
    "action": "RobotAction",
    "endpoint": "/Robots/Conveyor/",
    "data": { "forward": false, "on": true
        }
  },
  ... -- more operations
  ]},
... -- more device types here
]
```

C. Implementation & Technical Realization

To evaluate our approach, we developed a prototype application that builds upon web technologies, specifically Node.js and React.js to build the proposed Speech IO Framework. For the speech recognition, we use a client-side, browser-based speech-to-text API[8] that is easy-to-use and offers solid speech recognition capabilities.

The mapping of spoken text to actual commands is based on word-by-word matching of texts, testing if the commands' keywords, and any device, operation or attribute names/aliases are present in the transcripts of the voice input. After matching the recognised transcript against the above commands on a "first-match" basis, the application performs the required REST calls to the DT API and provides speech-based user feedback using the browser's speech output functionality. This DT API is implemented in phyton, using a timescale database for storing historical attribute values. Note, that we are currently working towards a more powerful text analysis, as is common in AI-based voice assistants.

D. Demonstration

We applied our speech interaction framework to the two before presented cases. The code of the speech interface is

[8]https://www.npmjs.com/package/react-speech-recognition

available in our research lab's repository: https://github.com/cdl-mint/SpeechInterface_DigitalTwins.

Furthermore, readers are invited to watch a video of the speech interface in action: https://youtu.be/7u45YXOH-jc.

IV. Discussion

Our (informal) evaluations with colleagues show that offering a speech-based alternative to classical dashboards or control panels can lead to more natural interactions and higher user satisfaction. Providing command aliases allows users to further customize how they use their systems.

Evidently, the prototype presented in this paper is an initial development and hence still suffers from certain limitations. Although we are actively working towards removing those, we provide a short discussion of some of them here:

Word-based speech matching The current command-matching is based on simple tokenisation of input text into words and matching it to device, operation and attribute names/aliases. Ideally, our approach should be extended to integrate the capabilities known from modern, AI-based voice assistants and speech recognition.

Custom speech interaction models As our research focuses on the speech framework itself, we defined a custom, JSON-based serialization of UML diagrams that we could easily adapt and extend. We are currently evaluating different ways to bring this information to "standardised" UML formats using, e.g., UML profiles.

Behavioural model data At the moment, our approach only extracts information from structural models. We are, however, exploring the possibility of also integrating behavioural models such as sequence and activity diagrams, which would enable dynamic adaptation of speech interfaces based on the system's state.

Larger sytems Our initial proof-of-concept has been applied to exemplary systems in our lab. To test the applicability of our framework generation, we are planning to generate speech interfaces for large systems with hundreds of device types and devices. We expect that the generation of speech interfaces for such systems will entail new scalability challenges.

V. Related Work

Ever since the development of speech interfaces, there has been vast interest in enabling this form of interaction in various domains as an alternative to screen/keyboard/mouse inputs. In recent years, with the growth of IoT applications, the amount of research in this field is growing strongly. For instance, to name just two interesting applications, [18] explored an overview of speech-based techniques for coding and IoT applications, and [19] provides an overview of voice-activated devices in health care applications. Next to these, many large tech companies provide their own voice assistants and connect them to, e.g., smart home applications or extend car entertainment systems with voice controls.

Due to the vastness of the explored applications, we narrow down the related work to the modelling domain. The probably closest to our work is the research by Peltonen [20] and Lahtinen [21], who derive speech commands from UML models. The former explicitly uses voice to interact with UML class diagrams, e.g., for the creation of objects and changing of attribute values. The latter work extends voice interface generation to include state charts and state machines for the control flow. This is similar to our approach, where we consider an expansion of our approach to include behavioural models as future work.

There are also approaches for voice-based modelling. Dana et al. explore *voice-driven modelling* [22], while Lahtinen and Peltonen worked on integrating speech commands to UML tools [23], [24].

To the best of our knowledge, though, none of the existing approaches target DTs or DT systems, or look into adding speech interface support to existing legacy systems or RESTful APIs.

VI. Conclusion

This paper describes a novel approach to enabling speech-based interaction for digital twin (DT) systems. Specifically, our approach is based on the generation of voice commands based on information extracted from structural system models such as UML object and class diagrams. In the course of development, we identify three extensions to standard UML notation that are required to enable a more capable speech framework configuration: First, the use of "aliases" for device, operation and attribute names enables the use of a more natural way of speaking. Secondly, we note that especially for systems such as ours, it is essential that the UML models can be extended with API information to also support legacy APIs that have not yet been updated to the new system architecture. Finally, we identified the need for "custom actions" which enables an additional pre- and post-processing of information before/after the call to the DT's API.

Based on the initial work presented in this paper, we plan to integrate more powerful, AI-based speech processing to replace the word-token-based command matching. Additionally, we also aim to attach our framework to standardised formats and tools such as EMF/Ecore and the Eclipse ecosystem. Finally, we feel motivated to enable further interaction capabilities and speech interaction from other types of models (e.g., behavioural ones such as UML sequence or activity diagrams).

Acknowledgement

Financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG) is gratefully acknowledged.

References

[1] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018.

[2] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, and M. Wimmer, "Digital twin platforms: requirements, capabilities, and future prospects," *IEEE Software*, vol. 39, no. 2, pp. 53–61, 2021.

[3] J. Pfeiffer, D. Lehner, A. Wortmann, and M. Wimmer, “Modeling capabilities of digital twin platforms-old wine in new bottles?” *J. Object Technol*, vol. 21, no. 3, p. 3, 2022.

[4] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, Second Edition*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.

[5] J. Hirschberg and C. D. Manning, “Advances in natural language processing,” *Science*, vol. 349, no. 6245, pp. 261–266, 2015.

[6] B. Denby, T. Schultz, K. Honda, T. Hueber, J. M. Gilbert, and J. S. Brumberg, “Silent speech interfaces,” *Speech Communication*, vol. 52, no. 4, pp. 270–287, 2010.

[7] F. Tao, B. Xiao, Q. Qi, J. Cheng, and P. Ji, “Digital twin modeling,” *Journal of Manufacturing Systems*, vol. 64, pp. 372–389, 2022.

[8] P. Muñoz, J. Troya, and A. Vallecillo, “Using UML and OCL models to realize high-level digital twins,” in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*. IEEE, 2021, pp. 212–220.

[9] G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, “Digital twin data modeling with automationml and a communication methodology for data exchange,” *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 12–17, 2016.

[10] D. Lehner, S. Sint, M. Vierhauser, W. Narzt, and M. Wimmer, “AML4DT: a model-driven framework for developing and maintaining digital twins with AutomationML,” in *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 1–8.

[11] 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.

[12] J. Michael, I. Nachmann, L. Netz, B. Rumpe, and S. Stüber, “"Generating Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines",” in *Modellierung 2022*. GI, 2022, pp. 33–48.

[13] P. Tonella and A. Potrich, “Reverse engineering of the UML class diagram from C++ code in presence of weakly typed containers,” in *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*, 2001, pp. 376–385.

[14] H. S. Govindasamy, R. Jayaraman, B. Taspinar, D. Lehner, and M. Wimmer, “Air Quality Management: An Exemplar for Model-Driven Digital Twin Engineering,” in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion Proceedings*. IEEE, 2021, pp. 229–232.

[15] D. Lehner, S. Sint, M. Eisenberg, and M. Wimmer, “A pattern catalog for augmenting Digital Twin models with behavior,” *Autom.*, vol. 71, no. 6, pp. 423–442, 2023.

[16] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.

[17] A. Colantoni, A. Garmendia, L. Berardinelli, M. Wimmer, and J. Bräuer, “Leveraging Model-Driven Technologies for JSON Artefacts: The Shipyard Case Study,” in *IEEE/ACM 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021.

[18] T. Bäckström, “Speech Coding, Speech Interfaces and IoT - Opportunities and Challenges,” in *52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1931–1935.

[19] P. Spachos, S. Gregori, and M. J. Deen, “Voice Activated IoT Devices for Healthcare: Design Challenges and Emerging Applications,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 7, pp. 3101–3107, 2022.

[20] J. Peltonen, S. Lahtinen, and K. Koskimies, “A specification technique for model based derivation of speech interfaces,” in *IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 2004, pp. 251–253.

[21] S. Lahtinen, H. Suontausta, and K. Koskimies, “Automated Derivation of Speech Interfaces: A Model-Based Approach,” in *19th Australian Conference on Software Engineering (ASWEC)*. IEEE, 2008, pp. 289–299.

[22] D. Black, E. J. Rapos, and M. Stephan, “Voice-Driven Modeling: Software Modeling Using Automated Speech Recognition,” in *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion*, 2019, pp. 252–258.

[23] S. Lahtinen and J. Peltonen, “Enhancing usability of UML CASE-tools with speech recognition,” in *IEEE Symposium on Human Centric Computing Languages and Environments*, 2003, pp. 227–235.

[24] ——, “Adding speech recognition support to UML tools,” *Journal of Visual Languages & Computing*, vol. 16, no. 1-2, pp. 85–118, 2005.