Integration of micro-services as components in modeling environments for low code development

Hafiz Ahmad Awais Chaudhary
CSIS, University of Limerick, Limerick, Ireland
Confirm - Center for Smart Manufacturing
ahmad.chaudhary@ul.ie

Tiziana Margaria
CSIS, University of Limerick, Limerick, Ireland
Lero - The Irish Software Research Center
Confirm - Center for Smart Manufacturing
tiziana.margaria@ul.ie

Abstract—**Low code development environments are gaining attention due to their potential as a development paradigm for very large scale adoption in the future IT. In this paper, we propose a method to extend the (application) Domain Specific Languages supported by two low code development environments based on formal models, namely DIME (native Java) and Pyro (native Python), to include functionalities hosted on heterogeneous technologies and platforms. For this we follow the analogy of micro services. After this integration, both environments can leverage the communication with pre-existing remote RESTful and enterprise systems' services, in our case Amazon Web Services (AWS) (but this can be easily generalized to other cloud platforms). Developers can this way utilize within DIME and Pyro the potential of sophisticated services, potentially the entire Python and AWS ecosystems, as libraries of drag and drop components in their model driven, low-code sytle. The new DSLs are made available in DIME and Pyro as collections of implemented SIBs and blocks. Due to the specific capabilities and checks underlying the DIME and Pyro platforms, the individual DSL functionalities are automatically validated for semantic and syntactical errors in both environments.**

Index Terms—**Domain Specific Language (DSL), Model Driven Development (MDD), eXtreme Model Driven Development (XMDD), Service Independent Building Blocks (SIBs), Low code development environments, DIME, Pyro.**

I. Introduction

Low code development platforms enable their users to design and develop applications with minimal coding knowledge [1], with the support of drag-and-drop visual interfaces that operate on representations of code as encapsulated code wrappers. The main aim [2] of these platforms is to produce flexible, cost effective and rapid applications in a model driven way. Ideally, they are adaptive to enhancements and less complex is terms of maintenance. Model-driven development (MDD) is an approach to develop such systems using models and model refinement from the conceptual modelling phase to the automated model-to-code transformation of these models to executable code [3]. The main challenges with traditional software development approaches are the complexity in development at large scale, the maintenance over time, and the adaptation to dynamic requirements and upgrades [1]. Doing this on source code is costly, and it systematically excludes the application domain experts. who are the main knowledge and responsibility carriers. At the same time, the cost of quality documentation and training of new human resources for code-based development are other concerns in companies and organizations that depend on code.

Domain Specific Languages (DSLs) conveniently encapsulate most complexities of the underlying application domain. Encapsulation of code and abstraction to semantically faithful representations in models empowers domain experts to take advantage of these platforms. They can develop products in an efficient manner and also meet the growing demands of application development without having deep expertise in software development. Based on a study [4] from 451 researches, the maintenance effort with low code platforms proved to be 50-90% more efficient as compared to changes with classical coding languages.

Software systems in general, and especially web apps in internet-centered ecosystems and digital threads in an Industry 4.0 context, are not isolated in nature: they demand interaction with various external systems, libraries and services. Frequent needs are (but not limited to)

- acquire sensors data from external systems,
- feed data to external dashboards for analytics and publishing,
- utilize the compute power of cloud systems,
- reuse sophisticated enterprise services.

In this context, microservices [5] play an important role at the enterprise level. The microservices paradigm (SOA done right) defines certain methods to design software services as suite of independently deployable components with the purpose of modularity, reusability and autonomy [5]. Different versions of these services may coexist in a system as a set of loosely coupled collaborative components and must be independently replaceable without impacting the operations of heterogeneous systems.

This paper proposes the integration of microservices as components in two graphical modelling development environments based on formal models: the general purpose, desktop DIME [6] Integrated Modelling Environment and the special purpose, web based Pyrus(Pyro) [7]. Their extension and integration with external systems through services extends the capabilities of these platform to meet wider communication needs (e.g. in the cloud), and also to take advantage of existing sophisticated enterprise services (e.g. AWS).

Low-code programming both at the API and the platform level is considered to be a game changer for the economy of application development. Gartner Inc., for example, predicts [8] that the size of the low-code development tools market will increase by nearly 30% year on year from 2020 to 2021, reaching a \$5.8 billion value in 2021. They state that so far, this is the fastest and probably the simplest and most economical method of developing applications.

In this paper, Sect. II discusses the state of art, Sect. III states the problem, Sect. IV gives an overview of the platforms used to extend the low-code DSLs. Sect. V explains the integration, architecture and implementation of SIBs in DIME (the desktop IME) and blocks in Pyro/Pyrus (the web IME). Finally, in Sect. VI we conclude and discuss.

II. State of the art

Most domain specific languages today are at the coding level and do not leverage a model driven approach at the platform level. The rise in re-usability and maintainability demands paved the path to low code development environments and gained the attention of the developer's community [9]. The construction of meta-models behind these DSLs is challenging, since they must capture all the domain knowledge, i.e. provide both semantic and syntactic rules. Ktrain [10] is a popular coding level DSL: a python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of python code. Xatkit [11], still in early stages of development, increases the reusability of chat bots by evolving NLP/NLU engine for text analytics. At the language level they support several versions of bots, but the generation of chatbots from existing data sources at the framework level is in future plans. jABC [12] is a general purpose XMDD framework for the development of desktop and enterprise applications in model driven fashion. It enables its users to compose models by drag and drop of reusable blocks into hierarchical graph structures that are executable (interpreted) and compilable. Aurera [1] is a standalone desktop system for business modelling and addresses the challenges of frequent changes to IT solutions. The system is in early stages of development and does not support communication with external systems. DIME [6] is a general purpose MDD platform-level tool, suitable for agile development due to its rapid prototyping for web application development. It follows the One Thing Approach based on XMDD [13], in a lineage of development environments that traces back to the METAFrame'95 [14]. DIME supports both control flow and data flow modelling in its process diagrams. Control flow models admit a single start node but may have multiple end nodes, and nodes (called SIBs)representing single functionalities or sub-models are graphs, i.e. formal models. The SIBs are connected via directed edges depending on the business logic, with distinct edge types for dataflow and control-flow. Agent-based modelling paradigm [15] is another popular approach to increase the development productivity in simulation environments. CaaSSET [16] is a Context-as-a-Service based framework to ease the development of context services. The transformation into executable services is semi-automatic.

The market segment of web based development environments is still relatively young. Not having many established environments, there is a huge potential for research and collaboration in this area. Theia [17], is a textual DSL tool supporting both desktop and web based IDEs. Pyro [7] is a web base graphical modelling environment for the collaborative development of web applications based on DSLs. Pyro, like DIME, is itself a product modelled with the Cinco [18] Meta Tooling Framework, which provides a suite of textual DSLs in which to specify the models for which to generate editors. The MGL ("Meta Graph Language") defines the structural information on the tool's model; the "Meta Style Language" (MSL) file specifies the visual characteristics (e.g. shapes and colors) of this model. The "Cinco Product Definition" (CPD) file specifies the details of the tool generation. Both DIME and Pyro are advanced graph model editors generated in this way from Cinco specifications. In this sense, they share a common philosophy, the semantic and syntactic characteristics of their respective models and edit/check/manipulate capabilities are described formally in their MGL, MSL and CPD files.

To interact with external entities, Micro service [19] is a popular way to develop modular, reusable and autonomous service components. We adopt this approach to extend the functionalities of two of the platforms in a model driven way. Following the same principles of graphical microservices architectures, AjiL [20] is a good effort in this direction, but due to performance delays in complex applications, they shifted their focus from graphical to textual notations.

III. Problem Statement

We consider the DIME [6] and Pyro [7] Cinco-products as our case study. Both are graphical Integrated Modelling Environments for low-code/no-code application development, used to develop research [21], [22] as well as industrial applications. We will use DSLs to virtualize the technological heterogeneity of the services, delivering a simple, coherent and efficient extension to both low-code modelling platforms.

Concretely, we show how to extend the capabilities of the DSLs through new, heterogeneous services. We

1) extend DIME, an offline eclipse-based general-purpose MDD environment for Web applications, by integrating a generic RESTful service as a new component, technically adding a new executable SIB that a) represents and b) executes this REST service;
2) extend Pyrus/Pyro, a collaborative, web based special-purpose MDD environment for data analytics and AI/ML, by integrating cloud-based enterprise services in a similar fashion. Here we chose Amazon Web Services.

The models in the 2 IMEs are different: DIME has rich models that cover processes, data, GUI, roles and security, and supports both dataflow and control flow models. Pyrus is simpler, and supports only dataflow modelling, which is popular and suffcient in the analytics pipelines it addresses.

As the specific integration depends on the characteristic and expressive power of the models, there are differences.

The extension by integration adds to the tools the capability to communicate with sophisticated enterprise ecosystems, without sacrificing the flexible yet intuitive modelling style for the no-code users, who just use the DSLs that are available.

IV. Overview of the IMEs

Domain-specific languages aim at minimizing the domain/IT knowledge gap between domain experts and software developers by lifting the vocabulary, granularity and structure of the application domain into the modelling language, so that the modelling entities stay familiar to the domain experts and their intuition is indeed correct. Domain experts prefer graphical languages because of the haptic functionality of drag-and-drop from a collection of functionalities is an apt metaphor for the construction of complex behaviors from an appropriate network of identifiable, well understood building blocks along intuitive control flow and data flow patterns.

The effort to develop these tools from scratch is enormous. Consequently, the specialization and evolution of such tools is hindered by the sheer cost and complexity of managing their code and its quality and support. Cinco [18] was a game changer: a meta-level platform that wipes out this cost and complexity by providing the above described domain specific graphical modelling and code generation capabilities. Most Cinco products are based on Eclipse, enhanced with graphical modelling tools and various plug-ins. Suddenly, one can create a new Integrated Modelling Environment by specifying properties in three files and availing of the Cinco code generation capability for the target execution environment (e.g, eclipse or web). Modifications are not anymore at the code level: to change DIME or Pyro, one edits the specifying files and re-generates the tool with the appropriate generator.

In this paper[1], we will discuss the extension and integration of external systems as micro services in two of the Cinco's products DIME and Pyro (particularly Pyrus).

A. *DIME*

DIME is an Integrated Modeling Environment based on J2EE eclipse, to design, develop and deploy web applications in an agile paradigm. Its model types help users to graphically model and develop different aspects of ordinary web application: (i) data model, (ii) GUI model, (iii) business logic in terms of processes and persistence, and (iv) roles and security model. The specific functional capabilities are provided to the users as a family of Graphical DSLs. The GUI DSL and a DSL providing a collection of generic blocks (called SIBs, for Service Independent Building Blocks) come with DIME, and other, domain specific DSLs can be added at need. Modelling in DIME happens mainly by the mechanism of drag-and-drop of DSL components on a canvas, and components comprise a node and a predefined set of outgoing edges. DIME supports both data and control flow to implement different aspects of business logic. Consistency checks are built-in in the DIME MGL and MSL, so that errors are either prevented (e.g., an output cannot connect to another output) or detected (e.g. the model is incomplete because some edges are dangling, not connected). DIME follows the One Thing Approach philosophy [23] by enforcing the SIBs to be generic and encapsulate only the required functionality. This way, SIBs are easily understandable and reusable, and application experts that are not coders can develop complex applications by using the SIBs in the provided DSLs. GUI models represent single pages of the web application and links to the underlyong functionalities. Process models can be hierarchical, i.e. contain other process models, this way easing the organization and comprehension of the structure and behaviour of complex applications. Once the models are ready, the product generation step feeds the models collection to successive model-to-model and model-to-code transformers, resulting in a complete code generation for a standard web application runtime.

The setup for the development environment for DIME requires Java version 1.8 and eclipse dependencies to be installed on the development machine.

B. *Pyro*

In contrast to DIME, Pyro is a web based Cinco-product that runs in a web browser and turns it into a collaborative domain-specific graphical modelling environment for data-flow applications. Pyro stores objects and data types in a loosely coupled manner [24]. To incorporate the rich features of typical web application, like the built-in support of cross-platform and a reusable components focused architecture, its front-end is built upon the Angular Dart [25] framework. To meet the needs of uninterruptible user interaction with the modelling environment, data exchange is implemented via non-blocking REST-based asynchronous communication. As a more recent development, Pyro is being enhanced with performance optimization and integration of external systems.

Pyrus is a specific Pyro derivative specialized for dataflow models executing within the popular Jupyter notebook environment. It is therefore particularly attractive for data analytics and AI applications, that are frequerntly coded in Python.

Working with Pyro/Pyrus requires the platform deployment on a local or remote server accessible via browser.

V. Extending the IMEs

We show now how to extend DIME and Pyrus with RESTful services and cloud-based AWS services, respectively. This happens by implementing a new DSL consisting of a collection of capabilities that run on an external platform in a different technology. Effectively, these are akin to microservices. We show here exemplarily how to implement one such microservice for each case. The extension to other RESTful services or other AWS or similar services is then easy to achieve following these blueprints.

[1]The complete project code is available in Github: https://github.com/ahmadch1991/syrcose21

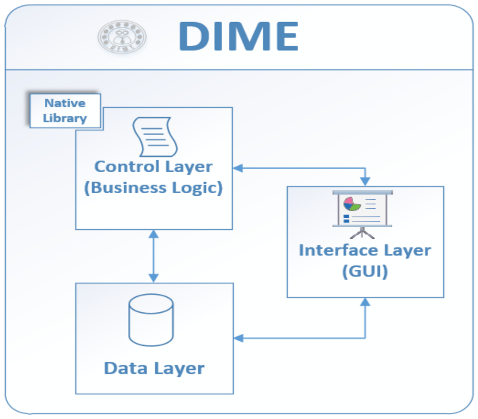


Fig. 1. DIME: Modelling Architecture and Native Library support

A. *RESTful extension of DIME*

We show now how to develop a generic Service Independent Building Block (SIB) in DIME that communicates with any external RESTful system.

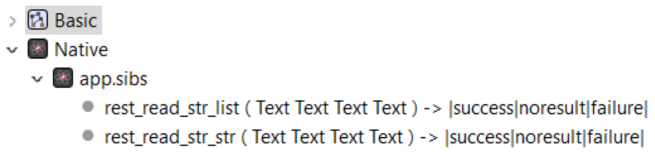
Extending the DIME functionality happens by using the support of native library it provides. In DIME's multi-model type architecture, the business/logic model type is where the new SIBs will be utilized. As shown in Fig. 1, the existing model architecture is extended with the addition of a native library as a new block belonging to the process/business logic model type. The native block will be merged to the process/business logic models during the automated code generation phase for the web application. Concretely, the extended functionality will be integrated as Java code with the remainder of the application during the compilation, and this way it will not add any additional performance penalty.

```
package app.demo
sib rest_read_str_list : file_path#Java_fn
    url : text
    input_var : text
    input : text
    output : text
    -> success
        output: [text]
    -> noresult
    -> failure
```

Listing 1. SIB declaration for the "REST Read" SIB

For the SIB implementation, we consider here a REST service that acts as a server and returns a list of country names on the basis of a country code input, e.g. United Kingdom for input 'uk', and the name of all countries from the database for input 'all'. The service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by client SIBs

Now, we need to create a new client SIB with appropriate characteristics to communicate with RESTful service. This encompasses the SIB declaration and the SIB implementation. The SIB declaration is shown in Listing 1.



```
> Basic
v Native
  v app.sibs
      rest_read_str_list ( Text Text Text Text ) -> |success|noresult|failure|
      rest_read_str_str ( Text Text Text Text ) -> |success|noresult|failure|
```

Fig. 2. SIBs explorer with the new Native SIBs

- Firstly, in the project explorer we add a new, empty file with extension ".sib" and the name of the proposed SIB.
- This SIB declaration file contains the signature of the new SIB. It starts with the keyword "sib", followed by the new SIB name, that in our case is REST_read_str_list, followed by a colon and the path to the attached Java function. This is the function be invoked when the SIB is used in the process modelling.
- The next section contains the proper signature: the list of inputs and outputs, with name and data types. In our case, the SIB accepts the following I/O:
  - URL of an external server
  - input variable name and data to create a valid URL at run time.
  - the output variable name is also added in the signature, to extract the requested data from server response for further JSON parsing.
- finally the list of different control branches based on outcomes. In our case the three branches are "success", which returns a text output provided by the external service, "noresult" of the external services returns no result, and "failure" in case of error in the communication with the external service.

For the SIB implementation, The RESTful "Rest Read" service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by this SIBs.

Once the declared SIB, its signatures and the attached Java function are validated by the platform, the SIB will be visible in the explorer as a Native SIB, with the other default SIBs as shown in Fig. 2. At this point it is ready to be used, and available to the DIME users as a drag and drop item, ready to be inserted in any process model.

Fig. 3 shows the visual representation of the newly developed SIB, as it appears when it is used in a process model. The required four inputs are being fed to this block using data flow (dotted) arrows. We see the three outgoing branches, labelled as defined. On success, the result will be conveyed as a string (or list of strings) to the successive SIB.

DIME automatically validates semantic and syntactic errors after the insertion and data connectivity of SIBs, ensuring this way the correctness of intended behaviour (automatic quality assurance [26] of models).

B. *Cloud extension of Pyrus*

We extend now the Pyrus is an online data analytics platform built using Pyro with Amazon Web Services (AWS),

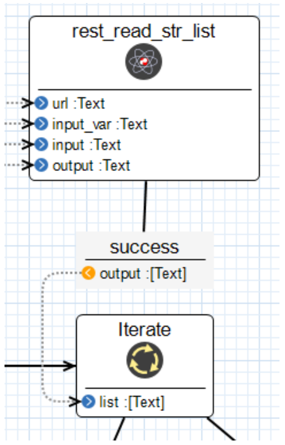


Fig. 3. The REST Read SIB in use: Visual representation in a model

choosing the Amazon Translate service [27]. Pyrus communicates with Jupyter hub at the backend. It uses the RESTful protocol to read function signatures and execute the attached python code. As shown in Fig. 4, Jupyter and Pyrus communicates in asynchronously manner.

The mechanism for the new AWS Translate block definition and implementation is similar to the DIME SIB declaration, but it only contains the signature, no outgoing branches. As Pyrus supports a dataflow modelling style, there are no control elements (the branches). The signature declaration starts with the # keyword, it is followed by meta data and the implementation of the functions in a python file. It has an extension ".py" and is located in the Jupyterhub space. Pyrus automatically reads these annotated signatures and shows them as drag-able blocks in its explorer.

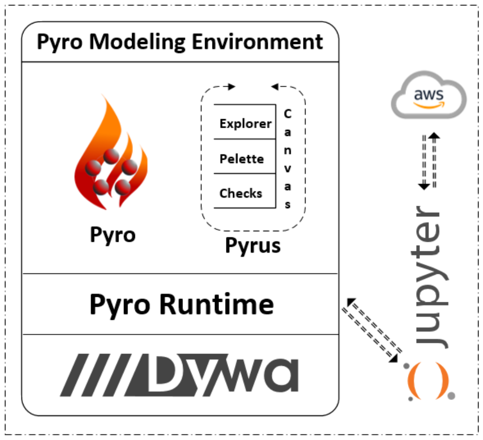


Fig. 4. The Pyrus/Pyro Architecture extended with AWS

Fig. 5 shows the working pipeline of AWS_translate: in reality we have defined 2 blocks, AWS init_session and AWS translate_string. The workflow is in fact logically divided in two phases: initialization and implementation.

The initialization block must meet the preconditions of the external server in order to use its services. Communication with the AWS server/services requires a valid session, validated with credentials, i.e. access key, secret key, server information. The "AWS.init_session" initiates the communication transaction with the AWS server. It accepts the required inputs/tokens from connected grey blocks, which are constants.

Once successfully authenticated by AWS, a session token is provided for further communication with AWS. This token (output) is fed to the "AWS.translate_string" block along with the other required inputs: the text string to be translated and the code of the from and to languages.

Finally, the (translated text) result is passed to the next block, "text_util.print_string", that prints it on screen.

The pipelines are automatically validated by the underlying modelling platform to check for connectivity errors of the blocks on the canvas.

C. *Tool and Technologies*

The tools and technologies used for these implementations and extensions are Eclipse, Java, JSON library, PHP, Python, DIME, Pyrus, Jupyter Hub and Amazon Web Services.

As the methodology is generic, it can be followed like a blueprint to implement communication and integrate a large variety of external services and platforms. The resulting drag and drop components enrich the DSL domain and expressive features of low code development in the mentioned platforms.

VI. Conclusion and Discussion

We presented a generic extension mechanism to two low code development environments along a microservice philosophy. We showed it by integrating preexisting remote RESTful services and cloud-based enterprise system services as new drag and drop components in the respective DSLs. In DIME, an offline low-code IME, we used the native library mechanism, with signature declaration, linked Java backend code, and the code is merged with the logic layer at compile time. Pyrus, an online no code graphical data analytics tool, is linked with Jupiter Hub for functions discovery and code execution. To display new python functions as components in Pyrus, custom signatures are added to the python files defined in Jupyter hub, and the data flow pipeline of the service is modelled in the Pyrus frontend.

The simplicity and generality of the integration are an important feature of the chosen platforms. We envisage in fact a systematic integration of DSLs for various application domains stemming for our research collaborations. The simpler this is, the easier is the adoption of the approach across diverse application domains, research groups, and industrial partners. The (hand)code based extension approach of most popular low-code environments, that do not use formal models, nor generate "intelligent" modelling domains that have built-in checks for the model conformance are in fact inferior and sources again of complexity in the management of heterogeneity, code maintenance and evolution. The next application domains will be data visualization and data streaming platforms. We will support more AI/ML and data analytics functionality both in DIME and in Pyrus, adding also cross-platform integration, in

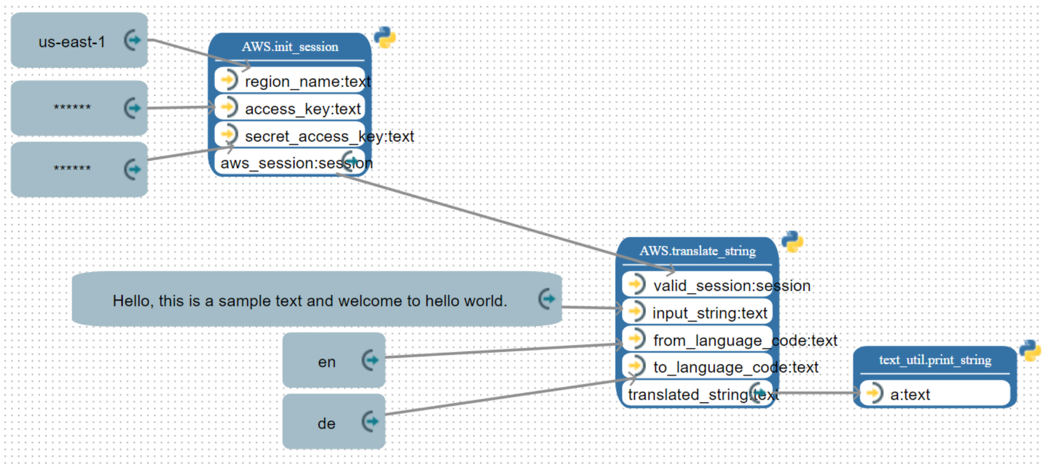


Fig. 5. Pyrus pipeline using AWS_translate

order to use the analytics capabilities of Pyrus pipelines in the DIME Digital Twin applications for Industry 4.0.

Acknowledgment

This work was supported by the Science Foundation Ireland grants 13/RC/2094 (Lero, the Irish Software Research Centre) and 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre).

References

[1] R. Waszkowski, “Low-code platform for automating business processes in manufacturing,” *IFAC-PapersOnLine*, vol. 52(10), pp. 376–381, 2019.
[2] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, “Low-code as enabler of digital transformation in manufacturing industry,” *Applied Sciences*, vol. 10, no. 1, p. 12, 2020.
[3] S. J. Mellor, T. Clark, and T. Futagami, “Model-driven development: guest editors’ introduction. ieee software, 20 (5). pp. 14-18. issn 0740-7459,” *IEEE software*, vol. 20, no. 5, pp. 14–18, 2003.
[4] (Accessed Feb, 2021) Intelligent process automation and the emergence of digital automation platforms. [Online]. Available: https://www.redhat.com/cms/managed-files/mi-451-research-intelligent-process-automation-analyst-paper-f11434-201802.pdf
[5] S. Newman, *Building microservices: designing fine-grained systems.* ” O’Reilly Media, Inc.”, 2015.
[6] 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 *ISoLA 2016*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2016, pp. 809–832.
[7] P. Zweihoff, S. Naujokat, and B. Steffen, “Pyro: Generating domain-specific collaborative online modeling environments,” in *Fundamental Approaches to Software Engineering*, R. Hähnle and W. van der Aalst, Eds. Cham: Springer International Publishing, 2019, pp. 101–115.
[8] (Accessed Feb, 2021) Gartner forecasts. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021
[9] K. Ordoñez, J. Hilera, and S. Cueva, “Model-driven development of accessible software: a systematic literature review,” *Universal Access in the Information Society*, pp. 1–30, 2020.
[10] A. S. Maiya, “ktrain: A low-code library for augmented machine learning,” *arXiv preprint arXiv:2004.10703*, 2020.
[11] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, “Xatkit: A multimodal low-code chatbot development framework,” *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.
[12] B. Steffen, T. Margaria, R. Nagel, S. Jörges, and C. Kubczak, “Model-driven development with the jabc,” in *Hardware and Software, Verification and Testing*, E. Bin, A. Ziv, and S. Ur, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 92–108.
[13] T. Margaria and B. Steffen, *eXtreme Model-Driven Development (XMDD) Technologies as a Hands-On Approach to Software Development Without Coding*, A. Tatnall, Ed. Cham: Springer International Publishing, 2020.
[14] B. Steffen, T. Margaria, A. Claßen, and V. Braun, “The metaframe’95 environment,” in *CAV*, R. Alur and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 450–453.
[15] F. Santos, I. Nunes, and A. L. Bazzan, “Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation,” *Simulation Modelling Practice and Theory*, vol. 104, p. 102126, 2020.
[16] H. Moradi, B. Zamani, and K. Zamanifar, “Caasset: A framework for model-driven development of context as a service,” *Future Generation Computer Systems*, vol. 105, pp. 61–95, 2020.
[17] (Accessed Feb, 2021) Cloud and desktop ide platform. [Online]. Available: https://theia-ide.org/
[18] S. Naujokat, M. Lybecait, D. Kopetzki, and B. Steffen, “Cinco: a simplicity-driven approach to full generation of domain-specific graphical modeling tools,” *International Journal on Software Tools for Technology Transfer*, vol. 20, pp. 1–28, 06 2018.
[19] L. Baresi and M. Garriga, “Microservices: The evolution and extinction of web services?” *Microservices*, pp. 3–28, 2020.
[20] F. Rademacher, J. Sorgalla, P. Wizenty, S. Sachweh, and A. Zündorf, “Graphical and textual model-driven microservice development,” in *Microservices*. Springer, 2020, pp. 147–179.
[21] T. Margaria and A. Schieweck, “The digital thread in industry 4.0,” in *International Conference on Integrated Formal Methods*. Springer, 2019, pp. 3–24.
[22] S. Jorges, C. Kubczak, F. Pageau, and T. Margaria, “Model driven design of reliable robot control programs using the jabc,” in *Proc. EASe’07*, vol. 07, 2007, pp. 137–148.
[23] T. Margaria and B. Steffen, “Business process modeling in the jabc: the one-thing approach,” in *Handbook of research on business process modeling*. IGI Global, 2009, pp. 1–26.
[24] J. Neubauer, M. Frohme, B. Steffen, and T. Margaria, “Prototype-driven development of web applications with dywa,” in *ISoLA 2014*, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 56–72.
[25] (Accessed Feb, 2021) Angular dart open source packages. [Online]. Available: https://github.com/angulardart
[26] S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer, “Active continuous quality control,” in *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, 2013, pp. 111–120.
[27] (Accessed Feb, 2021) Amazon translate; fluent and accurate machine translation. [Online]. Available: https://aws.amazon.com/translate/