Design of a Model Library for the Low-Cost Functional Development of Mechatronic Systems

Sven Jacobitz*, Jie Zhang, Xiaobo Liu-Henke

Institute for Mechatronics, Ostfalia University of Applied Sciences, Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Germany; *sve.jacobitz@ostfalia.de*

SNE 34(4), 2024, 215-223, DOI: 10.11128/sne.34.tn.10714
Selected ASIM WS 2023 Postconf. Publication: 2023-10-15
Rec. Improved English V.: 2024-08-12; Accepted: 2024-09-20
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The increasing complexity of mechatronic systems demands a structured and systematic approach to their development, thereby highlighting the importance of Rapid Control Prototyping (RCP) as an essential methodology.
Consequently, it is imperative to have a comprehensive Computer Aided Engineering (CAE) platform to facilitate support throughout the development process. Traditional CAE platforms, however, are often prohibitively expensive, leading to the development of LoRra at Ostfalia as a low-cost alternative.
Central to this paper, the LoRra model library supports the entire RCP development process through robust data management that includes advanced version and configuration management capabilities. By utilizing a systematic, hierarchical organization of models within a tree-like structure and enabling multi-user access, the library promotes consistent and traceable development processes. This framework not only enhances the reusability and reliability of development artifacts but also lowers barriers to adopting rapid control prototyping methodologies for small and medium-sized enterprises (SMEs).
This paper discusses the design, functionality, and implementation strategies of the LoRra library, demonstrating its potential to significantly influence the future of mechatronic system development by improving accessibility and systematic management.

Introduction

The complexity and functional scope in mechatronic systems is constantly increasing. This trend represents a major challenge for small and medium-sized enterprises (SMEs). To remain competitive, they must integrate more and more intelligent hardware and software into their products. This can be attributed to the increasing number of functions, as well as the growing number of complex and interdependent software components [1].

A structured, systematic development of such systems is indispensable to handle ever shorter development times with higher quality requirements [2]. Holistic model-based Rapid Control Prototyping (RCP) methodology that is frequently employed in this context. It is imperative that a Computer Aided Engineering (CAE) platform provides seamless support in order to achieve a high degree of automation, which is a fundamental aspect of RCP. The cost of established CAE platforms represents a significant barrier to the introduction of the RCP process, particularly for small and medium-sized enterprises (SMEs). As part of the EU-funded research project *Low-Cost Rapid Control Prototyping-System with Open-Source Platform for the Functional Development of Embedded Mechatronic Systems (LoCoRCP)*, the low-cost development platform LoRra was created at Ostfalia [3].

In order to guarantee the reusability of models and functions, a systematic approach to data management is employed, encompassing associated sub-processes such as version and configuration management. This is indispensable due to the high diversity, flexibility, and short lifetime [4]. The initial regulatory framework for this was established at NASA in the early 1960s, as documented in [5]. As Sax et Al. [6]observe, inconsistencies in function development are increasingly caused by the high number of variants. This issue can be avoided by using appropriate configuration management. In the context of RCP, this requires a CAE-based model library that manages all relevant artifacts (such as models, program source code, or documentation) [7].

Version and configuration management are widely utilized in the domain of classical software development. For an overview, see [8]. One of the most commonly utilized open-source tools in versioning is GIT [9]. Nevertheless, the primary objective of this tool is to facilitate the change-based management of text files [10].

Nevertheless, the application of this change-based approach to data formats commonly utilized in RCP is not particularly practical, as evidenced by [11]. Consequently, it is essential to implement suitable adaptations for utilization in a model library.

To identify the requisite adaptations, Kruse and Shea conducted systematic investigations [7]. In this approach, a model is constructed in a standardized way from metadata, interface information, and parameters. Based on such a standardized structure, version and configuration management for complex, composite models can also be carried out using a system entity structure (SES) [12].

In the following, the CAE-based LoRra model library is conceived, designed and realized as an example. The rest of the paper is structured as follows: Section 1 summarizes the RCP-development methodology and introduces the LoRra-platform. In section 2, the basics of the solution are designed based on an analysis of the requirements, which is then concretized into a design in section 3. Finally, a approach of realization is provided in section 4. Section 5 summarizes the results and provides an outlook for future work.

1 Development Methodology and Platform

Due to the high system complexity of modern interconnected mechatronic systems, the structured, model-based, verification-oriented RCP process is used for development and validation. This comprises the following process steps: modeling, analysis/synthesis, automated generation of source code, automated implementation on real-time hardware, and online experimentation. Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL) simulations are employed to support this methodology [13].

The methodology delineated in this work is distinguished by its high level of consistency and automation, encompassing modeling, model-based functional design, automated code generation, and real-time implementation, as depicted on the left side of Figure 1. This approach is supported by a fully automated, seamless CAE platform. Specifically, the LoRra development platform represents a modular, low-cost example of such a CAE system.

Figure 1 demonstrates the RCP development process and the comprehensive integration facilitated by LoRra (cf. [3]).

Central to this plattform is a model library that ensures a consistent and traceable development status throughout all phases of the development process.

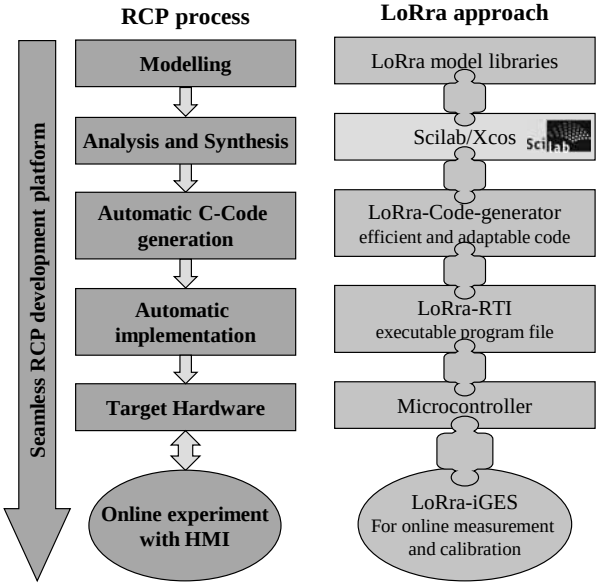

Figure 1: RCP development process with seamless support by the LoRra platform [3].

A comprehensive multi-domain model library is utilized throughout the modeling process. This library facilitates the clear assembly and management of model variants through robust version and configuration management systems. The open-source CAE tool Scilab/Xcos is employed for the analysis and synthesis of functions, offering functionalities comparable to those found in the commercially frequently utilized Matlab/Simulink.

Subsequently, the resultant functional model can be seamlessly integrated into the model library. The LoRra Application Programming Interface (API)'s open interfaces enable straightforward incorporation of additional modeling programs and interface drivers, enhancing the system's versatility. MiL simulations are leveraged to optimize and validate the functions at an early developmental stage.

The LoRra code generator implements model-to-text transformation techniques to automatically produce efficient, modular C source code derived from the functional model. This generator is designed with open functional descriptions of the model's basic elements, called basic blocks, allowing for flexible extensions.

The resulting source code can be reintegrated into the Xcos model for further optimization and testing via SiL simulations without the need for manual intervention [14].

Upon achieving a satisfactory functional status, the interfaces with the controlled system models or other functions are substituted with blocks from the LoRra Real-Time Interface (RTI). This substitution facilitates the configuration and programming of real-time hardware without manual coding.

The hardware-specific RTI framework includes a real-time operating system and standardized interface drivers, enabling automated deployment on real-time hardware platforms such as low-cost microcontrollers, for instance, those from the STM32H7 series. HiL simulations are employed to optimize and evaluate the developed functions under real-time conditions [15]. Additionally, the integrated graphically-supported experimentation software (iGES) serves as an intuitive Human-Machine Interface (HMI) for controlling, monitoring, and recording data during online experiments.

2 Conception of the Library

This section is dedicated to the concept of the model library and begins with an analysis of the requirements. Various development approaches for graphical user interfaces (GUIs) are then examined.

A basic solution approach is developed on the basis of these analyses. The detailed consideration of these aspects lays the foundation for the subsequent steps in the design and realization of the model library, described by the next sections.

2.1 Development Approaches for Graphical User Interfaces

Standardized architectural styles play a pivotal role in structured software design, facilitating reusability, organizing design processes, and establishing a uniform vocabulary [16]. Notably, over a quarter of these styles are employed in designing user interfaces [17]. The Model/View/Controller (MVC) principle is particularly significant in the context of this work.

Based on [18], the MVC architectural style is depicted in Figure 2. This framework segregates the visualization (View), control, and data model into distinct components with clearly defined interfaces. The control component responds to user interactions within the GUI and modifies the model as needed.

Additionally, the model may be altered by other software components. It communicates any modifications to the control component, enabling the latter to refresh the view. This architecture's low coupling between components makes it especially apt for HMIs that operate across various platforms [19]. For instance, the graphical visualization specific to an operating system can be entirely isolated from both the controller and the model, enhancing adaptability and maintainability.

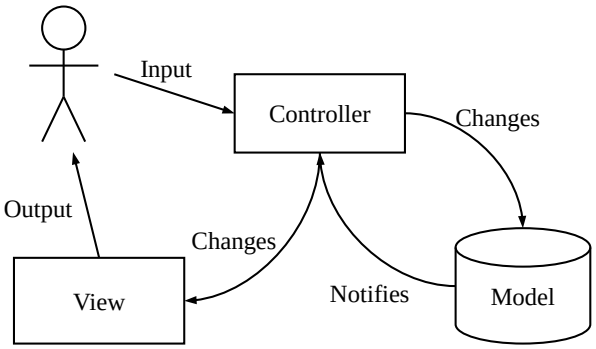

Figure 2: Idea of the architectural style Model/ View/Controler according to [18].

Applying his MVC principle ensures a modular reusable software design.

2.2 Requirements on the Model Library

As delineated in section 1, the model library serves as a crucial instrument for data management throughout the development process. To fully enhance version and configuration management, the model library must adhere to several stringent requirements:

1. **Version Control:** It must support comprehensive versioning of all data it contains, including functionalities for version management processes such as checking and releasing versions.
2. **Configuration Management:** The library should facilitate the necessary data structures for configuration management across all stages of the development process, along with associated processes (e.g., checking, release) to maintain consistent data status throughout these stages.
3. **Hierarchical Organization:** Models should be categorically and hierarchically structured, allowing for configurable categories and levels that enhance organization and retrieval.

4. **Search Capability:** A robust search function is essential to efficiently locate specific models within the library.
5. **Collaboration Features:** The library should support collaborative work in distributed teams, enabling seamless access and modification of a shared model base.
6. **Comprehensive Display of Information:** It should provide a clear and detailed overview of all relevant model information to facilitate easy access and understanding for all users involved in the project.

These requirements are essential to ensure that the model library effectively supports the intricate and dynamic needs of modern software development environments.

2.3 Basics of the Solution

To fully address the requirements of the model library, it is imperative to examine the structure of a model in greater depth. The distinction between generic and aggregated models forms a crucial part of this analysis.

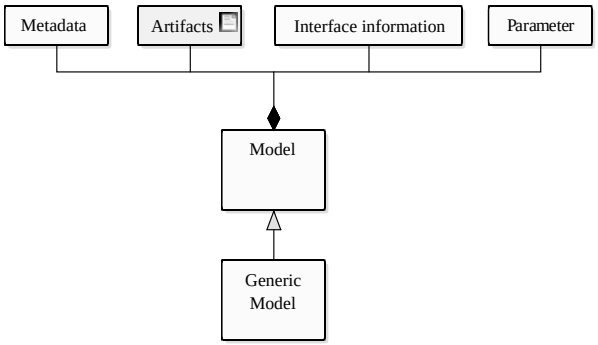

Figure 3: Structured setup of a generic model.

Generic Model: This represents the smallest coherent unit within the model library and operates at the lowest level of hierarchy. It is indivisible and does not contain any submodels structured hierarchically below it. An example of a generic model is the electrical part of a DC motor, represented by equation (1). Figure 3 visually details the structure of a generic model, which encompasses the following components:

$$u = Ri + L\frac{\mathrm{d}i}{\mathrm{d}t} - u_i \tag{1}$$

- **Metadata:** These attributes detail overarching characteristics such as the model's name, author, and a general description.
- **Interface information:** This includes the data structure, units, and other pertinent details about the model's inputs and outputs. For instance, using equation (1), the terminal voltage u in volts (V) as input and the motor current i in amperes (A) as output.
- **Parameters:** Specifications and values for the model's parameters, such as the resistance R in ohms (Ω) and the inductance L in henries (H).
- **Artifacts:** These may include the simulation file (e.g., in Xcos format), generated C code, or documentation associated with the model.

Aggregated Model: This type of model comprises multiple sub-models and represents a higher hierarchical level within the library. Aggregated models are structured as configurations in the model library, formed by integrating specific versions of part models. The concept of assembling these configurations is illustrated in Figure 4.

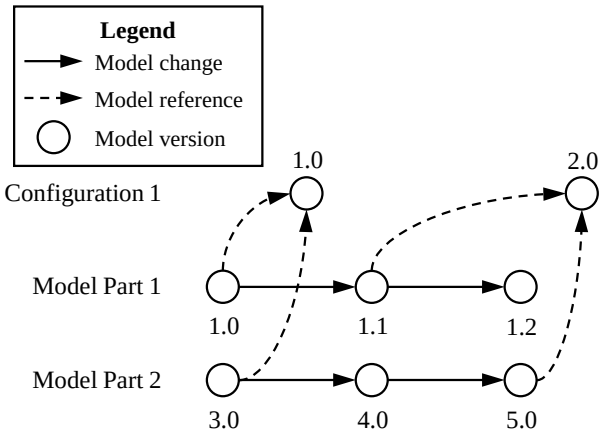

Figure 4: Basic concept of assembling a configuration.

In terms of organization, models should be structured hierarchically in a tree-like format within the library. This structure includes folders (grouping hierarchy element) and model elements (both generic and aggregated models). User rights management is crucial, allowing permissions to be assigned both to groupings and individual model elements.

To facilitate multi-user access, a central storage principle is utilized, as shown in Figure 5. A local working copy is employed to access and modify model artifacts. Any changes made are then synchronized back to the central repository, which functions as a comprehensive database.

Following synchronization, users can update their local copies with the modified data, ensuring that all team members have access to the most current versions of models and configurations.

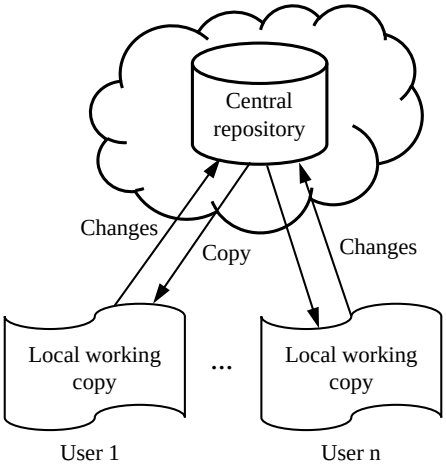

Figure 5: Concept for the central storage of models.

Employing this approach enables the systematic design of a model library seamlessly integrated into the RCP process. This integration facilitates efficient and coherent management of model data throughout the development cycle.

3 Design of the Library Functions

Building on the concepts introduced in section 2, this section provides a detailed elaboration of the proposed system.

Initially, we examine the data structures and interfaces, as well as the mechanisms for data management. Subsequently, the design of the GUI is addressed to ensure a seamless and efficient user experience. This structured approach ensures that both the backend and frontend components of the system are robustly developed and integrated.

3.1 Data Structures

The design of the model library requires intricate data structures and interfaces to support its functionality. At the core of the library is the hierarchical model tree, which also serves as the foundational database for the GUI developed according to the MVC principle. This section elaborates on the design of the data structure and interfaces that underpin the model tree.

The data structures are designed using object-oriented principles. The abstract class *AModelElement* acts as the foundational structure for each element within the tree, encapsulating essential data such as the element's title, its path within the tree, and its parent element.

Derived from *AModelElement* are two key classes: *HierarchyElement* and *ModelElement*.

- **HierarchyElement:** This class manages a collection of subordinate elements, effectively organizing them within the tree structure to facilitate navigation and management.
- **ModelElement:** This class is more specialized and includes detailed interface information, parameters, and storage details of the model, among other pertinent data. This encapsulation ensures that each model within the library is both self-contained and richly described for easy access and modification.

Figure 6 provides a visual representation of these relationships through a UML class diagram, illustrating how these elements are interconnected to form a robust and scalable model tree structure. This structure is critical for supporting the complex interactions and data management requirements of the model library in the RCP process.

The object-oriented design of the data structures ensures that the model library is both scalable and adaptable, supporting complex hierarchies and detailed model management. This architectural choice enhances functionality and integration within the RCP process, facilitating efficient navigation and modification of models.

3.2 Data Management

The data management within the model library primarily hinges on robust version and configuration management systems.

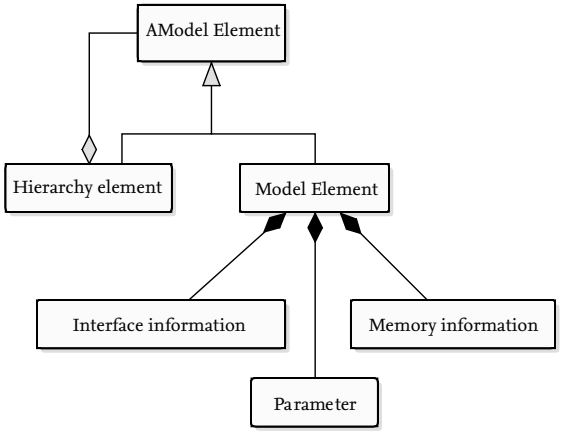

Figure 6: UML diagramm of the hierarchical model tree.

To facilitate consistent versioning and thereby enable structured reuse in configurations, the library must support a version management process that includes the formal release of new versions. Modifications to a model necessitate its reincorporation as a new version following company-specific approval procedures. For the LoRra model library, versions suggested by users are only made publicly available after obtaining consent from designated groups, as mandated by organization-specific protocols.

Version identification within the model library utilizes a semantic versioning approach by the numerical format, x.y, where 'x' represents the major version and 'y' the minor version.

Minor version increments occur when updates (such as bug fixes) do not alter model compatibility (i.e., behavior and interfaces remain unchanged). Conversely, major version increments – accompanied by resetting the minor version to zero – are employed when changes impact model compatibility, such as modifications to interfaces or functionality enhancements. Models are integrated into configurations by referencing these version numbers, as depicted in Figure 4, which outlines this versioning principle.

Effective data management within the model library is achieved through meticulous version and configuration management, ensuring that models are consistently updated and released in alignment with defined organizational processes. This systematic approach supports the seamless integration and reuse of models across various configurations, enhancing the library's utility and reliability.

3.3 Graphical User Interface

The GUI of the LoRra model library is designed in accordance with the MVC principle, as outlined in section 2.1.

The foundation for this design (the model component) is the hierarchical model tree developed in section 3.1, complemented by various control classes tasked with specific functions.

Figure 7 depicts the comprehensive layout of the GUI. The main window is segmented into three primary parts:

- **Navigation Area:** This section hosts the hierarchical model tree, enabling users to perform various actions such as opening or editing models, and provides an initial overview of the current model status. It also includes a search function for navigating the model tree.
- **Display Area:** This part offers different views for the presentation and modification of information, including metadata, model artifacts, and version history.
- **Tool Area:** This section features a context-sensitive toolbar and a menu structure designed to facilitate the operation of the library.

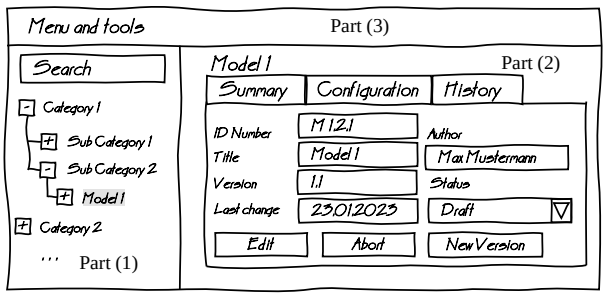

Figure 7: Draft of the model library GUI.

This structured approach ensures a clear and effective interaction with the model library, enhancing user experience and operational efficiency.

4 Realization

The initial implementation of the model library utilizes Java within an object-oriented Eclipse Rich Client Platform framework (cf. [20]).

Listing 1: Exemplary model tree in JSON format.

```
{
  "title" : "root",
  "relPath" : "",
  "children" : [ {
    "title" : "Vehicle models",
    "relPath" : "vehicle_models/",
    "children" : [ ... ]
  }, {
    "title" : "Function models",
    "relPath" : "function_models/"
       ,
    "children" : [ {
      "title" : "VMS",
      "relPath" : "VMS/",
      "children" : [ ... ]
    },
    {
      "title" : "AMS",
      "relPath" : "AMS/",
      "children" : [ {
        "relPath" : "efm/",
        "metaFileName" : "efm.json
           ",
        "repoUrl" : "https://
           tinyurl.com/repo_efm/"
      }, ... ]
    } ]
  }, ... ]
}
```

The Eclipse platform provides numerous built-in mechanisms essential for development, including the *Standard Widget Toolkit* and event-driven, low-coupling communication among various graphical components. Moreover, it supports a wide range of extendable plug-ins with open interfaces.

Version management is integrated using the open-source tool GIT (cf. [9]), which offers established versioning protocols. The LoRra setup facilitates the avoidance of the limitations identified earlier by employing structured, text-based model descriptions.

Initially, user authentication is set up for the Atlassian service Bitbucket, with potential for future enhancements.

The model descriptions are structured in JSON format, aligning with standards outlined in [21]. Listing 1 illustrates an example of a model tree stored in this format.

Hierarchy elements are designated by attributes such as *title* (the display title of the element), *relPath* (the relative file path to the parent hierarchy element), and *children*. Model elements are characterized by fields like *relPath*, *metaFileName*, and *repoUrl* (the URL to the online GIT repository), with all pertinent metadata stored in the file identified by *metaFileName*.

The integration of sub-models into configurations is managed through XML-based SES, facilitating the establishment of an abstract structure for new configurations as a preliminary phase. This is followed by the generation of a specific configuration by referencing the appropriate sub-models and specifying the required version and variant.

This implementation strategy leverages the robust capabilities of proven tools for efficient version management and flexible integration, ensuring a scalable and modular architecture. By utilizing open text based formats for structured data representation, the model library effectively supports complex configurations and facilitates seamless data management and access.

5 Summary and Outlook

The paper discusses the design and implementation of a model library within the Low-Cost Rapid Control Prototyping platform LoRra, aimed at enhancing the functional development of embedded mechatronic systems. The LoRra library offers a cost-effective alternative to traditional CAE platforms, particularly beneficial for SMEs. It incorporates systematic data management with robust version and configuration management to ensure reusable, reliable, and consistent access to development artifacts across the system's development cycle. The library utilizes a hierarchical, tree-like structure for organizing models and supports multi-user environments through central storage and local working copies, ensuring that all team members access the most recent data.

The model library's foundational framework is robust, but enhancements are planned to optimize functionality and user experience. These include adding undo/redo functions, improving navigation in the GUI, and integrating advanced graphical editors for better model configuration and visualization. Enhancing the GIT tool's differential capabilities for Xcos models will allow more effective visualization of changes, aiding in version tracking and management.

A new user authentication system will support integration with central storage, boosting collaboration across distributed teams.

Automated consistency checks will ensure model configurations are coherent before deployment, complemented by comprehensive documentation and training materials to help new users.

Performance will be optimized through algorithm improvements and efficient data handling, particularly for large-scale projects. Establishing a community for sharing models and best practices, along with collaborative development features like shared workspaces and real-time tools, will increase the library's utility and reach.

Focusing on these improvements, the LoRra model library aims to significantly enhance its capabilities and user satisfaction, supporting the advancement of mechatronic systems and adapting to user needs.

Acknowledgement

Funded by the Lower Saxony Ministry of Science and Culture under funding number ZN3495 in the Lower Saxony Advance of the Volkswagen Foundation and supervised by the Center for Digital Innovations Lower Saxony (ZDIN).

References

[1] Liu-Henke X, Scherler S, Fritsch M, Quantmeyer F. Holistic development of a full-active electric vehicle by means of a model-based systems engineering. In: *Proceedings of 2016 IEEE International Symposium on Systems Engineering (ISSE)*, edited by Rassa B, Carbone P. 2016; pp. 1–7. DOI: 10.1109/SysEng.2016.7753142.

[2] Jacobitz S, Göllner M, Zhang J, Yarom OA, Liu-Henke X. Seamless validation of cyber-physical systems under real-time conditions by using a cyber-physical laboratory test field. In: *Proceedings of the International Conference on Recent Advances in Systems Science and Engineering (RASSE)*. IEEE. 2021; pp. 1–8. DOI: 10.1109/RASSE53195.2021.9686844.

[3] Jacobitz S, Liu-Henke X. The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering. In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*. SciTePress. 2020; pp. 57–64. DOI: 10.5220/0008993500570064.

[4] Kittlaus HB. *Software Product Management*. Berlin, Heidelberg: Springer. 2022. DOI: 10.1007/978-3-662-65116-2.

[5] Fahmy S. The Evolution of Software Configuration Management. *International Journal of Advanced Trends in Computer Science and Engineering*. 2020; 9(1.3):50–63. DOI: 10.30534/ijatcse/2020/0891.32020.

[6] Guissouma H, Klare H, Sax E, Burger E. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, edited by IEEE. 2018; pp. 298–305. DOI: 10.1109/SEAA.2018.00056.

[7] Kruse B, Shea K. Design Library Solution Patterns in SysML for Concept Design and Simulation. *Procedia CIRP*. 2016;50:695–700. DOI: 10.1016/j.procir.2016.04.132.

[8] Ratti N, Kaur P. Case Study: Version Control in Component-Based Systems. In: *Designing, Engineering, and Analyzing Reliable and Efficient Software*, edited by Singh H, Kaur K, pp. 283–297. IGI Global. 2013;DOI: 10.4018/978-1-4666-2958-5.ch016.

[9] Eriksson H, Sun J, Tarandi V, Harrie L. Comparison of versioning methods to improve the information flow in the planning and building processes. *Transactions in GIS*. 2021;25(1):134–163. DOI: 10.1111/tgis.12672.

[10] Nugroho YS, Hata H, Matsumoto K. How different are different diff algorithms in Git? *Empirical Software Engineering*. 2020;25(1):790–823. DOI: 10.1007/s10664-019-09772-z.

[11] Schmitz D, Deng W, Rose T, Jarke M, Nonn H, Sanguanpiyapan K. Configuration Management for Realtime Simulation Software. In: *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE. 2009; pp. 229–236. DOI: 10.1109/SEAA.2009.69.

[12] Durak U, Pawletta T, Oguztuzun H, Zeigler BP. System entity structure and model base framework in model based engineering of simulations for technical systems. In: *Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering*, edited by D'Ambrogio A, ACM Digital Library. Society for Computer Simulation International. 2017; pp. 1–10. DOI: 10.22360/springsim.2017.mod4sim.001.

[13] Liu-Henke X, Jacobitz S, Scherler S, Göllner M, Yarom O, Zhang J. A Holistic Methodology for Model-based Design of Mechatronic Systems in Digitized and

Connected System Environments. In: *Proceedings of the 16th International Conference on Software Technologies (ICSOFT)*, edited by Fill HG, van Sindern M, Maciaszek L. SciTePress. 2021; pp. 215–223. DOI: 10.5220/0010566702150223.

[14] Jacobitz S, Liu-Henke X. Automatic Code Generation for a Seamless Low-cost Development Platform. In: *10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, edited by Pires LF, Hammoudi S, Seidewitz E. 2022; pp. 294–301. DOI: 10.5220/0010894300003119.

[15] Jacobitz S, Liu-Henke X. A Real-Time Interface for Xcos – Demonstrated on a Battery-management System. In: *2nd Scilab Conference*. 2019; pp. 1–8.

[16] Bass L, Clements P, Kazman R. *Software architecture in practice*. SEI series in software engineering. Upper Saddle River NJ u.a.: Addison-Wesley, 3rd ed. 2013. ISBN: 978-0321815736.

[17] Henninger S, Corrêa V. Software pattern communities: current practices and challenges. In: *Proceedings of the 14th Conference on Pattern Languages of Programs*, edited by Aguiar A, Yoder J. ACM Press. 2007; pp. 1–19. DOI: 10.1145/1772070.1772087.

[18] Adams S. MetaMethods: The MVC paradigm. *HOOPLA!* 1988;1(4).

[19] Liu Z, Li F, Liu H, Wu C, Zhang J. A Study of Cockpit HMI Simulation Design Based on the Concept of MVC Design Pattern. In: *Proceedings of the 3rd International Conference on Modelling, Simulation and Applied Mathematics (MSAM 2018)*. Atlantis Press. 2018; pp. 82–84. DOI: 10.2991/MSAM-18.2018.19.

[20] Vogel L. *Eclipse Rich Client Platform: The complete guide to Eclipse application development*. Hamburg: Vogella, 3rd ed. 2015. ISBN: 978-3943-747133.

[21] International Organization for Standardization. ISO/IEC 21778:2017: Information technology - The JSON data interchange syntax. 2017.