A Model Library Tool for Holistic Embedded Software Design

Sven Jacobitz, Xiaobo Liu-Henke
Ostfalia University of Applied Sciences
Department of Mechanical Engineering, Institute for Mechatronics
Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Germany
Email: {sve.jacobitz; x.liu-henke}@ostfalia.de

***Abstract*—The ever-increasing complexity and connectivity of mechatronic systems makes using a structured, systematic methodology essential for embedded software design. The model-based Rapid Control Prototyping is a widely used model-based development process for this. Essential is a seamless support by a Computer Aided Engineering platform. However, such platforms are very cost-intensive, which is why the seamless low-cost platform LoRra was developed by the authors. A key element of this platform is the Model Library tool, which provides consistent access and traceable change management to all data (especially functional and plant models as well as resulting artefacts) used throughout the holistic process of software development. Version and configuration management also increase the reusability of resulting artefacts. This paper presents the new ideas, used to design the LoRra model library for the low-cost function development of mechatronic systems. The holistic coverage of the entire development process, from modelling to real-time realization, is the feature, that distinguishes the LoRra model library from existing tools.**

***Index Terms*—Rapid Control Prototyping (RCP), low-lost development platform, model-based design, model library.**

I. Introduction

Mechatronic systems continue to increase in complexity and functionality. This trend is a major challenge for Small and Medium-sized Enterprises (SMEs). To remain competitive, they need to integrate ever more intelligent hardware and software into their products. This is not only due to the number of functions in a system, but also to the ever-increasing degree of connectivity between complex software components that strongly interact with each other [1]. The structured and systematic development of such software intensive systems is essential to meet ever shorter development times and higher quality requirements [2]. In this context, model-based Rapid Control Prototyping (RCP) is a widely used methodology for embedded software design. The seamless support by a Computer Aided Engineering (CAE) development platform is essential for RCP in order to achieve a high degree of automation. Established seamless CAE tool chains are very cost-intensive, which is a major barrier to the adoption of the RCP process, especially for SMEs [3]. Therefore, as part of the EU-funded research project *Low-Cost Rapid Control Prototyping System with Open-Source-Platform for Functional Development of Embedded Mechatronic Systems (LoCoRCP)*, the authors developed a seamless low-cost development platform named LoRra [4].

This paper presents the conception, design and exemplary realization of the CAE-based LoRra model library. This library enables access to a consistent data base throughout the entire development process, as well as traceable change management - especially for functional or plant models and resulting artefacts. The rest is structured as follows: Section III summarizes the RCP development methodology and introduces the LoRra platform. The state of the art is outlined in Section II. In Section IV, the concept and the basic solution approach is presented based on a requirement analysis. The design is detailed in Section V. Section VI is a description of the implementation. Finally, Section VII summarizes the results and gives an outlook on future work.

II. State of the art

Developing in distributed teams and the associated central data management has been an important topic in classical software development for a long time. This is especially, due to the high diversity, flexibility and short development times of software [5]. Model-based software development poses new challenges for methods and tools. To ensure the reusability of models and the associated software functions newly, systematic, integrated data management approaches with the associated sub-processes, such as version and configuration management must be used. This is important because, in comparison to classical software development, the resulting artefacts no longer result from manual textual changes, but are derived from models [6].

First regulations for this came up in the early 1960s at NASA [7]. According to Sax et al. [8], inconsistencies during function development are increasingly caused by the high number of variants, which can be avoided by using an appropriate configuration management. In the context of RCP, a CAE-based model library that manages all relevant artefacts (result of a subprocess such as models, program source code or documentation) is recommended for this purpose [9].

Version and configuration management is widely used in classical software development. An overview is provided by [10]. A primitive but widely used method is manual versioning. A backup is generated by manual copying and renaming. In comparison, the backup copies are created automatically when version control tools are used. There are many

 ISBN: 978-1-68558-042-1

different kinds of software available, such as the Concurrent Version System (CVS), Subversion (SVN) or Mercurial.

A frequently used open source tool for versioning is GIT [11]. However, the focus of this tool is on change-based management of text files [12]. An application of this approach to data formats common in model-based design is not very practicable [13]. Therefore, extensive adaptations are required for use in a model library.

In order to identify the type and scope of the necessary adaptations, systematic investigations were carried out by Niedzwiedz and Frei [14], for example. Here, a model is constructed in a standardized way from metadata, interface information and parameters. Based on such a standardized structure, version and configuration management can be performed even for complex, integrated models. An example of such a systematic structural description approach is the System Entity Structure (SES) [15].

In summary, there is currently no applicable solution for central model management within the framework of a low-cost RCP platform. Available approaches are either designed for textual changes, but do not offer the structures necessary for model-based software design or do not support essential processes, such as version and configuration management. Approaches that exist for UML-based models, for instance (c.f. [16]), often only support the models themselves, but not the resulting artefacts such as the source code. Also, the holistic coverage of the entire development process, from modelling to real-time realization, is not supported by other tools, yet.

III. 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 software development and validation. This consists of the process steps modelling, analysis / synthesis, automated generation of source code, automated implementation on real-time hardware and online experimentation. The whole methodology is supported by Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) simulations [17].

The presented methodology is characterized by a high degree of consistency and automation, from modelling and model-based software design to automatic code generation and real-time realization (cf. Fig. 1 on the left). It is accompanied by a seamless, fully automated CAE platform. The modular, cost-effective development platform LoRra is such a CAE platform. Fig. 1 illustrates the RCP development process, as well as the seamless support by means of LoRra [4]. Of particular relevance here is a central model library that makes a consistent, traceable development status available in all process steps.

The domain-independent model library serves as a central data base from the modelling process up to the realization. By means of version and configuration management, model variants can be designed, managed and integrated to higher level models. The open source CAE tool Scilab / Xcos (cf. [18]) is used for model analysis and synthesis of the

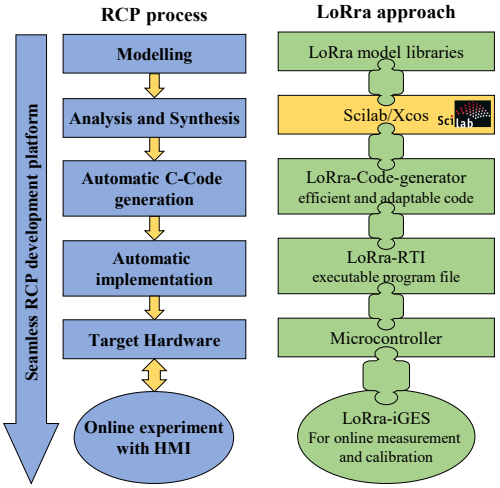

Fig. 1. RCP development process with seamless support by the LoRra platform [4].

software. It offers a wide range of functionality comparable to those of the commercially frequently used Matlab / Simulink. The resulting function model can be integrated directly into the model library. Thanks to the open interfaces of the LoRra API, existing programmes and interface drivers can also be integrated with little effort. MiL simulations can be used to optimize and test the developed functions at an early stage of development.

The LoRra code generator automatically generates efficient, modular C source code from the functional model by means of model-to-text transformation. Open functional descriptions of basic elements of the model, so-called basic blocks, make the LoRra code generator flexibly extendable. The generated source code can be re-integrated into the Xcos model without manual work, e.g., for optimization and testing by means of SiL simulations.

The signals to the plant models or, depending on the development focus, to other software components are replaced by interface blocks of the LoRra Real-Time Interface (RTI) when the development reaches a sufficient functional status. This enables the use of real-time hardware interfaces without manual programming. In combination with hardware-specific RTI basic software, which includes a real-time operating system and standardised interface drivers, automated implementation on the real-time hardware by the RTI is possible. Low-cost microcontrollers, e.g., of the STM32H7 series, are used as real-time hardware. By means of HiL simulations, the developed function can thus also be optimized and tested under real-time conditions. The integrated Ggraphics-supported Experimentation Software (iGES) is available as a Human-Machine Interface (HMI). It can be used to intuitively perform and monitor online experiments, as well as to record measurement data by means of real-time data acquisition.

IV. Concept of the Model Library

In this section, the conception of the LoRra model library is presented. First, some approaches for the graphical user

 ISBN: 978-1-68558-042-1

interface development are introduced. Then, the requirements are outlined and the initial stage for a solution is derived.

A. Human Machine Interface development approaches

Nowadays, standardised architecture styles are used for structured software design [19]. These serve in particular to increase reusability, to structure the design and to create a uniform vocabulary. More than 25% of the existing styles are used for the design of HMI [20]. In the context of this work, the Model-View-Controller (MVC) principle is particularly relevant.

The architectural style MVC, according to [21], is illustrated in Fig. 2. Here, the view (also called visualization or presentation), the controller and the data model are realized separately with defined interfaces. The controller component reacts to user inputs in the Graphical User Interface (GUI) and changes the model if necessary. Furthermore, the model can also be changed by other software components. It notifies the controller of the changes made. The controller updates the presentation. Due to the low component coupling, this principle is particularly suitable for HMI that are used on different target platforms [22]. For example, the operating system-dependent graphical presentation can be completely decoupled from the controller and the model.

B. Model library requirements

As outlined in Section III, the model library is a central tool for data management in all process steps of software development. In order to fully support the model-based version and configuration management approach, the model library must meet the following overarching requirements:

1) Coherent versioning of all contained data and support for version management processes (e.g., review, approval).
2) Supporting the data structures required for Configuration Management throughout the process steps, as well as the Configuration Management processes (e.g. review, approval) to ensure a consistent data state at all times and across all RCP steps.
3) Hierarchical structuring of the models in configurable categories and hierarchy levels.
4) Search function to quickly find specific models, even though a large amount of data are contained.
5) Support for distributed teams working from a common model base.
6) Presentation of all relevant model information in one overview.

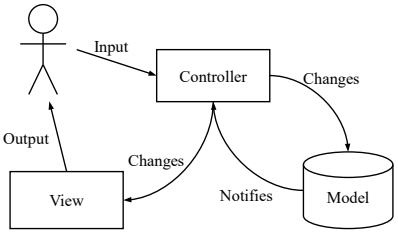

Fig. 2. Architecture style Model-View-Controller according to [21].

C. Basic idea of the concept

In order to be fully compliant with the requirements of the model library, the first step is a closer look at the structure of a model. In doing this, generic and aggregated models need to be distinguished.

A generic model is the smallest self-contained model unit at the lowest hierarchical level. It is not subdivided into further hierarchically ordered sub-models. An example of a generic model is the electrical part of a DC motor, which can be described by (1) (cf. [23]). The structured assembly of a generic model is illustrated by Fig. 3. It consists of four components:

- **Metadata** describes the higher-level characteristics (e.g., name, author, general description) of the model.
- **Interface information:** Data structure, units and other relevant information of the inputs- and outputs of the model. Using the example of (1), the terminal voltage u in V and the angular speed ω in rad/s as input or the motor current i in A as output.
- **Parameter:** Information and values about the parameters of the model. Using (1) as an example, the resistance R in Ω, the inductance L in H and the machine constant c in Vs.
- **Artefacts** of the model such as the (Xcos-)model file, the generated C-code or the model documentation.

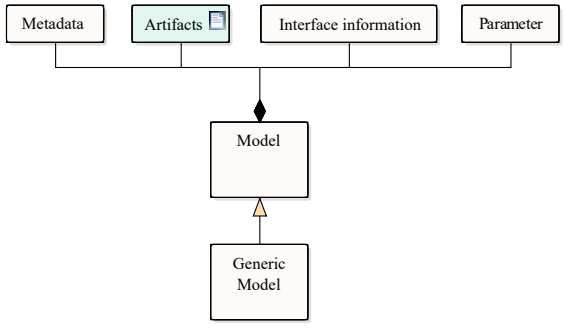

Fig. 3. Structured assembly of a generic model.

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

An aggregated model is composed of further sub-models and thus represents higher hierarchy levels. Aggregated models are mapped as so-called configurations in the model library. A configuration is created by integrating defined version levels of the part models. Fig. 4 illustrates the principle assembly of a configuration.

The models are arranged hierarchically in a tree structure. The tree contains folders (grouping hierarchy elements) and model elements (both generic and aggregated models). User rights and individual processes can be assigned to both groupings and individual elements.

To enable model access for several users, the principle of a central data repository is applied. Fig. 5 illustrates the concept. All model data are stored in the central repository, which acts

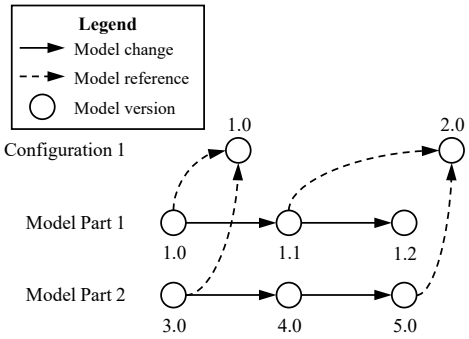

Fig. 4. Principle assembly of a configuration.

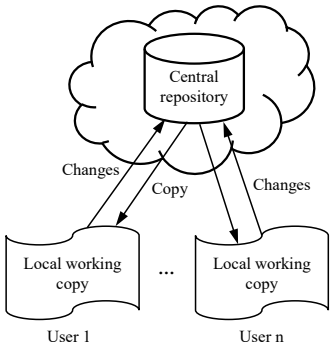

Fig. 5. Concept of the central data repository.

as a database. Model artefacts are accessed via local working copies. Changes are transmitted to the central repository and merged to the original data. Users can then pull the changed data to their local working copy.

V. Model library design

The concept from Section IV will now be fleshed out and transferred into a concreted design. For this purpose, the data structures and interfaces, as well as the data management are designed.

A. Data structures

There are a number of data structures and interfaces that are necessary for the model library. The core element is a hierarchical model tree, which also serves as the data basis for the MVC of the GUI. In the following, the data structure and interfaces of the model tree are designed as an example.

The set-up of the data structure is object-oriented. For each element of the tree, the abstract class *AModelElement* represents the basic structure. It contains central data such as title, path in the tree or parent element. The classes *HierarchyElement* and *ModelElement* are derived from it. *HierarchyElement* contains a list with subordinate elements. *ModelElement* summarizes interface information, parameters and memory information of the model among other data. Fig. 6 illustrates the relationship as a UML class structure.

The model tree requires interfaces for various operations, which are served by the controller classes. For example, adding or moving child elements of the class *HierarchyElement* is

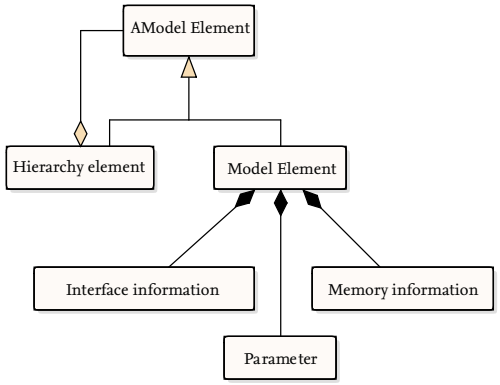

Fig. 6. Class structure of the hierarchical model tree.

provided here. Model elements require more extensive interfaces, for example to change the metadata or to generate new versions. Each modification must be captured and documented by the version management.

B. Data Management

Data management is the key function of the model library. On the one hand, version and configuration management are very important. On the other hand, the storage of local working copies must also be coordinated. For example, only models that have been selected by users should be completely downloaded as working copies. For other models, storing the metadata is enough.

To ensure versioning and thus consistent reuse of software in the form of configurations, the model library must support an appropriate version management process. This mainly concerns releasing new versions. If changes are made to a model or other artefacts, a new model version may only be used following specific release processes. For the LoRra model library, this means that versions proposed by users are not released to the public until they have been approved by the groups of people required according to the configured process.

For version numbering, the concept of semantic versioning is applied. Different versions of a models and artefacts are identified by version numbers of the form x.y. Where x is the major version and y is the minor version. If no compatibility-relevant adjustments were made to the model during a change (e.g., bug fixes or pure visual changes - behaviour and interfaces remain the same), only the minor version is incremented. If adjustments have been made that affect the compatibility of the model with other models (e.g., changes to the interfaces, extension of the functionality), the major version is incremented and the minor version is set to 0.

The composition of a configuration is done by linking submodels. For this purpose, the corresponding version numbers of the models are referenced, and the interfaces are linked. Fig. 4 illustrates the principle.

 ISBN: 978-1-68558-042-1

VI. Realization

The model library is implemented in Java as Eclipse Rich Client Platform (cf. [24]). A basic set of functions is implemented in an object-oriented way. The Eclipse framework already offers many mechanisms necessary for realization, such as the *Standard Widget Toolkit* or event-based, minimal-coupling communication between different graphical elements. In addition, numerous extensions with open interfaces can be used.

Versioning is done using the existing open source tool GIT (cf. [11]), which also connects to the central storage infrastructure. There is a separate GIT repository for each model. Proven mechanisms for versioning are already available here. By using structured, text-based model descriptions, the limitations mentioned in Section II can be avoided. The GIT branching enables variant management in addition to the version and configuration management functions described above. Initially, user authentication is implemented for the Atlassian service Bitbucket. Later extension is possible.

The structured model description is in JSON format (cf. [25]). Listing 1 contains an exemplary stored model tree. Hierarchy elements are identified by the fields *title* (display title of the element), *relPath* (relative file path to the parent hierarchy element) and *children*. Model elements contain the fields *relPath*, *metaFileName* and *repoUrl* (URL to the online GIT repository). All relevant metadata is stored in the file specified in *metaFileName*.

The integration of sub-models into a configuration is XML-based. This is done in the form of an SES. Thus, the structure of a new configuration can first be created at an abstract level. A concrete configuration is then generated by pruning and referencing the sub-models and specifying the version and variant. This approach with flexible, standardised data structures and interfaces allows the model library to be applied in various simulation environments. It can therefore be used as part of the LoRra platform, as well as in the context of other development platforms or further model editors.

Finally, the GUI is built according to the MVC principle introduced in Section IV-A. The data basis for this (model) is the model tree designed in Section V-A. Fig. 7 illustrates the overall design (presentation) of the LoRra model library. The main window is divided into three areas. The navigation area (on the left) contains the hierarchical model tree of the library. Here, users can perform actions on individual models (e.g., open or edit) and get an initial overview of the current model status. In addition, the model tree can be searched. The display area (on the right) contains various views for displaying and editing information. Here, for example, the metadata and model artefacts can be displayed or the version history can be viewed. In addition, a context-dependent toolbar and the menu structure for operating the library are arranged in the tool area.

VII. Summary and Future Work

This paper presents the design of a model library for low-cost software development of mechatronic systems using model-based design approaches. As part of the seamless RCP development platform LoRra, which is based on open source software, the model library provides a consistent and traceable model basis for each development step. In this way, the model

Listing 1. Exemplary model tree in JSON format.

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

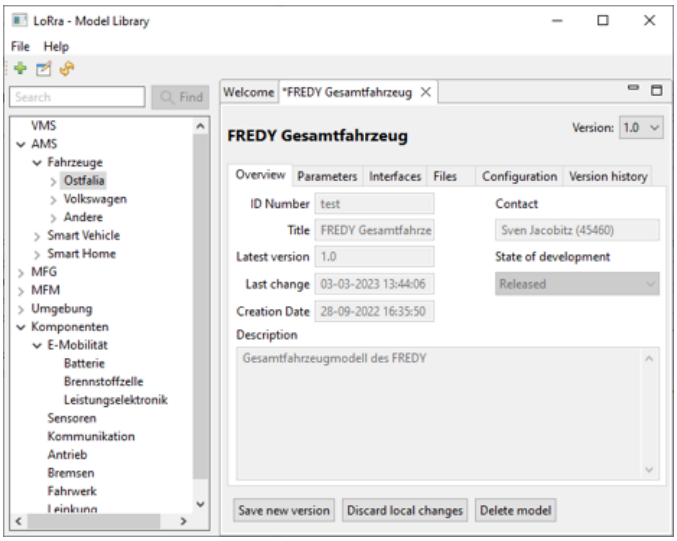

Fig. 7. Layout of the graphical user interface of the model library.

library covers the entire development process holistically. Based on the basic requirements, an approach for version and configuration management of hierarchical models, as well as for central model storage was developed. This was followed by the exemplary design of a model tree data structure for hierarchical model configurations and the graphical user interface. Finally, the implementation of a basic functionality is summarized.

Based on existing technologies, such as version management with GIT, a new tool for the continuous development of software for mechatronic systems has been created. An easy-to-use graphical interface facilitates version and configuration management of project artefacts throughout the entire development process from MiL, SiL and HiL to prototype.

Future work will focus on further optimizing the user experience. To this end, the integration of graphical editors to simplify the generation and management of configurations is also possible. An extension of the GIT tool *diff*, which visualizes model changes, is planned for an optimized overview of the version history. Finally, a generalization of user authentication is possible, so that any kind of central storage system can be used. For further testing and optimization, the model library will be integrated into the virtual embedded software test bench of the authors.

ACKNOWLEDGMENT

Funded by the Lower Saxony Ministry of Science and Culture under grant number ZN3495 within the Lower Saxony "Vorab" of the Volkswagen Foundation and supported by the Center for Digital Innovations (ZDIN).

REFERENCES

[1] X. Liu-Henke, S. Scherler, M. Fritsch, and F. Quantmeyer, "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)*, B. Rassa and P. Carbone, Eds., 2016, pp. 1–7.

[2] S. Jacobitz, M. Gollner, J. Zhang, O. A. Yarom, and X. Liu-Henke, "Seamless validation of cyber-physical systems under real-time conditions by using a cyber-physical laboratory test field," in *2021 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*. IEEE, 2021, pp. 1–8.

[3] X. Liu-Henke, R. Feind, M. Roch, and F. Quantmeyer, "Investigation of low-cost open-source platforms for developing of mechatronic functions with rapid control prototyping," in *Proceedings of the 2014 International Conference Mechatronic Systems and Materials (MSM)*, 2014, pp. 1–9.

[4] S. Jacobitz and X. Liu-Henke, "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 - Science and Technology Publications, 2020, pp. 57–64.

[5] H.-B. Kittlaus, *Software Product Management*. Berlin, Heidelberg, Germany: Springer, 2022.

[6] K. Henderson and A. Salado, "Value and benefits of model–based systems engineering (MBSE): Evidence from the literature," *Systems Engineering*, vol. 24, no. 1, pp. 51–66, 2021.

[7] B. L. Summers, "Software Configuration Management," in *Effective Methods for Software Engineering*, B. L. Summers, Ed. New York, USA: Auerbach Publications, 2020, pp. 57–65.

[8] H. Guissouma, H. Klare, E. Sax, and E. Burger, "An Empirical Study on the Current and Future Challenges of Automotive Software Release and Configuration Management," in *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 298–305.

[9] B. Kruse and K. Shea, "Design Library Solution Patterns in SysML for Concept Design and Simulation," *Procedia CIRP*, vol. 50, pp. 695–700, 2016.

[10] N. Ratti and P. Kaur, "Case Study: Version Control in Component-Based Systems," in *Designing, Engineering, and Analyzing Reliable and Efficient Software*, H. Singh and K. Kaur, Eds. Hershey, USA: IGI Global, 2013, pp. 283–297.

[11] H. Eriksson, J. Sun, V. Tarandi, and L. Harrie, "Comparison of versioning methods to improve the information flow in the planning and building processes," *Transactions in GIS*, vol. 25, no. 1, pp. 134–163, 2021.

[12] Y. S. Nugroho, H. Hata, and K. Matsumoto, "How different are different diff algorithms in Git?" *Empirical Software Engineering*, vol. 25, no. 1, pp. 790–823, 2020.

[13] D. Schmitz, W. Deng, T. Rose, M. Jarke, H. Nonn, and K. Sanguanpiyapan, "Configuration Management for Realtime Simulation Software," in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2009, pp. 229–236.

[14] S. Niedzwiedz and S. Frei, "A structured model library for the analysis of electric-vehicle drivetrains," in *AmE 2012 - automotive meets electronics*, ser. GMM technical report. VDE-Verlag, 2012, pp. 21–26.

[15] U. Durak, T. Pawletta, H. Oguztuzun, and B. P. Zeigler, "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*, A. D'Ambrogio, Ed. ACM Society for Computer Simulation International, 2017, pp. 1–10.

[16] R. S. Bashir, S. P. Lee, S. U. R. Khan, V. Chang, and S. Farid, "Uml models consistency management: Guidelines for software quality manager," *International Journal of Information Management*, vol. 36, no. 6, 2016.

[17] X. Liu-Henke, S. Jacobitz, S. Scherler, M. Göllner, O. Yarom, and J. Zhang, "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*, H.-G. Fill, M. van Sindern, and L. Maciaszek, Eds. SCITEPRESS - Science and Technology Publications, 2021, pp. 215–223.

[18] A. K. Verma and R. Verma, *Introduction to Xcos - A Scilab Tool for Modeling Dynamical Systems*, 1st ed. Jodhpur, India: MBM Engineering College, JNV University, 2020.

[19] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed., ser. SEI series in software engineering. Upper Saddle River, USA: Addison-Wesley, 2013.

[20] S. Henninger and V. Corrêa, "Software pattern communities: current practices and challenges," in *Proceedings of the 14th Conference on Pattern Languages of Programs - PLOP '07*, A. Aguiar and J. Yoder, Eds. ACM Press, 2007, pp. 1–19.

[21] S. Adams, "MetaMethods: The MVC paradigm," *HOOPLA!*, vol. 1, no. 4, 1988.

[22] Z. Liu, F. Li, H. Liu, C. Wu, and J. Zhang, "A Study of Cockpit HMI Simulation Design Based on the Concept of MVC Design Pattern," in *Proceedings of the 2018 3rd International Conference on Modelling, Simulation and Applied Mathematics (MSAM 2018)*. Atlantis Press, 2018, pp. 82–84.

[23] X. Liu-Henke, M. Gollner, M. Fritsch, R. Feind, and R. Buchta, "FreDy - An electric vehicle with intelligent chassis-control systems," in *2015 Tenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, 2015, pp. 1–8.

[24] L. Vogel and M. Milinkovich, *Eclipse Rich Client Platform: The complete guide to Eclipse application development*, 3rd ed., ser. Vogella series. Hamburg, Germany: Vogella, 2015.

[25] "ISO/IEC 21778:2017: Information technology — The JSON data interchange syntax," International Organization for Standardization.

 ISBN: 978-1-68558-042-1