Towards boosting the OpenMBEE platform with model-code consistency

Robbert Jongeling
Mälardalen University
Västerås, Sweden
robbert.jongeling@mdh.se

Antonio Cicchetti
Mälardalen University
Västerås, Sweden
antonio.cicchetti@mdh.se

Federico Ciccozzi
Mälardalen University
Västerås, Sweden
federico.ciccozzi@mdh.se

Jan Carlson
Mälardalen University
Västerås, Sweden
jan.carlson@mdh.se

ABSTRACT

Eventual consistency between design and implementation is imperative for the quality and maintainability of software systems. Towards achieving this consistency, engineers can analyze the gaps between models and corresponding code to gain insights into differences between design and implementation. Due to the different levels of abstraction of the involved artifacts, this analysis is a complex task to automate. We study an industrial MBSE setting where we aim to provide model-code gap analysis between SysML system models and corresponding C/C++ code through structural consistency checks. To this end, we propose an extension of the OpenMBEE platform, to include code as one of the synchronized development artifacts in addition to models and documentation. In this paper, we outline our initial research idea to include code as a view in this platform and we propose to explicitly link the code to generated documentation, and thereby to the model.

CCS CONCEPTS

• **Software and its engineering** → *Model-driven software engineering*; *Maintaining software.*

KEYWORDS

Consistency checking; Model-based systems engineering

ACM Reference Format:
Robbert Jongeling, Antonio Cicchetti, Federico Ciccozzi, and Jan Carlson. 2020. Towards boosting the OpenMBEE platform with model-code consistency. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3421409

1 INTRODUCTION

In model-based development (MBD) of complex software systems, a gap between abstract system representations and their concrete implementation is expected. MBD practices can take many forms, from informal whiteboard sketches to complete code generation from models. In the latter case, the gap between model and code is usually small, i.e. the model contains the same information as the code and the code can be created from the model by automated means. However, in typical industrial model-based systems engineering (MBSE), this gap is larger, i.e. the model contains a much more abstract representation of the system and the code is not supposed to be automatically generated from it.

Generally, it is understood that inconsistencies between disparate views of the same system are inevitable during development but they can be problematic if not identified in time [2]. To aid engineers in identifying possible inconsistencies between the model and code, we propose to introduce structural consistency checks. These checks can indicate structural differences between model and code that can be symptomatic of more serious lacks in the conformance of the code to the model. In particular, we propose to introduce model-code consistency checks in the Open Model-Based Engineering Environment (OpenMBEE) platform[1].

With this short paper, we aim to discuss with the OpenMBEE community the idea of extending the OpenMBEE platform to include code as a view. We argue that this extension would allow for simple structural consistency checks between the SysML system model and C/C++ code and providing engineers and managers with insights into model-code consistency.

2 MOTIVATION

The MBSE paradigm is applied in many different ways in industrial practice. Well-known standards guiding these practices are ISO42010 [5] and the INCOSE handbook for systems engineering [11], also commonly referenced is Friedenthal's "a practical guide to SysML "[4]. What follows in this section is a description of key characteristics of the MBSE setting as it is at our industrial partner. Although the specific setting serves as background and motivation for some of the design decisions we make in our proposed approach in Section 3, the approach does not depend on it.

2.1 An industrial MBSE setting

In broad strokes, the MBSE practice in the studied setting consists of: (i) the creation of a system model, (ii) the automatic generation of documentation from it, and (iii) the manual implementation of the system based on the generated documentation. The system model captures the decomposition of the system into components, as well as the interactions between those components. The entire process is a collaborative effort in which system engineers are responsible for designing features in the system model and communicating their designs to software engineers in handover meetings. The software engineers implement the design, based on the information in these

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3421409

[1] https://www.openmbee.org/

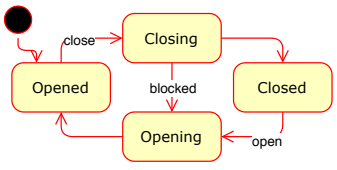
meetings and the generated documentation, but do not directly interact with the system model.

Despite this separation between the development of the system model and the code, the two artifacts should eventually be consistent with each other. Manually assessing this gap and the degree to which model and code are consistent becomes infeasible with the increasing size and complexity of systems. Therefore, automated means of guiding engineers to achieve eventual consistency, or at least alerting them of inconsistency, are valuable additions to the development process.

In the studied setting, the system model is the core development artifact capturing the decomposition of the system into several distinct function blocks. Those function blocks are further decomposed into system components, which represent units of functionality, each of which can be implemented as either software or hardware. In the remainder of this work, we focus on those system components that shall be implemented as software. These components are contained in software blocks in the system model. Their structure is captured in several block definition diagrams (BDDs) and internal block diagrams (IBDs). Activity diagrams are used to describe the behavior of methods and state machines are used to denote the life cycle of blocks, including possible interactions with other blocks. Furthermore, the interfaces between different components are defined using proxy ports. The system model defines both the communication protocol for interfaces and the messages that are to be sent over them.

2.2 Consistency checking

The described system model is refined into the implementation. In general, code and model can be inconsistent in terms of missing information in two directions: absence (elements in the model are not represented in the code) and divergence (elements in the code have no corresponding element in the model) [9]. It is worth noting that of course many detailed elements in the code are expected to not be represented in the model, so these divergences are limited to cases in which the elements would be expected to have a representation in the model. Furthermore, we consider contradictions between model and code as inconsistencies. As an example, consider the following code and the simple state machine of an automatic door in Figure 1.




```
void close();
void open();
/*absence:*/
// void blocked();
/*divergence:*/
void forceClose();
```

Figure 1: Simple state machine of an automatic door.

Based on the described MSBE practice in the studied setting, we identify several types of inconsistencies as relevant for our structural consistency checks to detect. We want to check rudimentary conformance of the code to the model. In particular, the following model elements should be represented in the code:

- Interface operations,
- Operations defined in state machines,
- Attributes of methods of system components, and
- Blocks in BDDs expected to be implemented as classes.

Symmetrical cases are of interest for ensuring the code does not diverge from the model. Checking for such divergences is challenging because many code elements by definition do not have model counterparts, because of the refinement relation between model and code. Therefore, we plan to check only consistency between method definitions in the code and their counterparts in the model. Contradictions can reveal themselves in several forms, e.g. interface violations, type violations, or even naming inconsistencies. If the names of explicitly linked model elements and code elements have a low degree of syntactic similarity, perhaps the link is outdated, or one of the elements should be updated.

Explicit correspondence links between model and code elements would allow us to analyze the artifacts with respect to the outlined structural consistency. Indeed, if we know which model and code elements should represent the same concepts, then the correctness and completeness of these links can be checked and possible inconsistencies indicated. Manual creation of such traceability links is often infeasible due to the size of the model and hence the number of required links. Therefore, we propose to: (i) limit the number of created traceability links, and (ii) simplify the creation and maintenance of them by allowing for their creation within the current development process. Since, in our setting, software engineers currently already base their implementation on the generated documentation, we expect the additional effort of manually linking code elements to the documentation to be small. Since the documentation is merely a generated view on the model, these links transitively also link the code to the system model, as depicted in Figure 2.

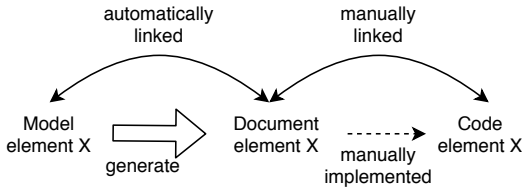


Figure 2: Proposed correspondence links are automatically generated between model and documentation, and manually created between documentation and code.

Our intent is to leverage the already established OpenMBEE platform and boosting it by enabling it to host these correspondence links and consistency checks. In it, multiple views on a system model can be synchronized across different tools by creating explicit correspondence links, or "cross-references" between different view elements. In Section 3, we introduce the current components of the OpenMBEE platform and our proposed extension for including code as a view on the system model.

3 THE OPENMBEE PLATFORM

As outlined, one of the challenges in the described MBSE setting is to obtain insights on the degree of synchronization between system model and code. More generally, there is a lack of synchronization

between different engineering artifacts, including documentation. The OpenMBEE platform provides components that build up a scalable and collaborative modeling environment in which models and documents are synchronized across different tools. This is achieved by considering the system model as the authoritative source of truth (AST) [10]. In this section, we explore the existing OpenMBEE components and the possibility of extending some of them to provide means for structural consistency checking across artifacts.

3.1 Existing OpenMBEE components

The core component of the OpenMBEE platform is the Model Management System (MMS). MMS contains a structured data representation of the system model and provides API functions through which the model can be inspected and manipulated. Furthermore, MMS serves as a version control system for this structured data. Branches can be created in the SysML tool and accessed through e.g. the view editor. Synchronization of branches and conflict resolution is supported through Model Development Kits (MDKs). MMS keeps track of revisions when views are updated.

Cameo Systems Modeler[2] is the supported SysML tool through the Cameo MDK. As part of the MDK, the OpenMBEE platform provides DocGen, which is a model-based approach for document generation [3]. To use DocGen, the engineer defines views and viewpoints within the system model that define the hierarchy and content of to-be-generated documents.

These generated documents, as well as any other document, may be edited in the View Editor (VE), which provides a web interface where different textual views of the model can be edited. VE further provides the engineer with the ability to define cross-references between model and document elements by using the MMS APIs. Once linked by such a cross-reference, elements are automatically synchronized across model and document.

The solid lines in Figure 3 show existing OpenMBEE components and their interactions. To provide our consistency checks, we propose to link the system model to documentation using the existing OpenMBEE tooling and further extend it to also link the code to model and documentation. In the next section, we outline our proposed usage and extension of OpenMBEE for providing model-code gap analysis and consistency checking.

3.2 A recipe for enabling consistency checking in OpenMBEE

In order to provide the links between model and code, as well as the structural consistency checks introduced in Section 2, we propose to treat code as yet another view of the system model. We utilize the existing relationship between model and generated documentation and provide additional means for engineers to relate the code they are developing to the generated documentation (and thereby, transitively, to the model). Finally, we utilize the MMS APIs to create structural consistency checks based on the created traceability links. The remainder of this section describes these ingredients, which are exemplified in Figure 3, using the same example state machine we saw earlier.

In the example, the consistency analysis script eventually notices the discrepancy between the model element (state machine) Door and the code element (class) Door. In Figure 3, the example output mentions the absence of a code implementation of the operation "blocked". Similarly, we anticipate the method could detect the divergence of the method "forceClose", which is present in the code but not in the model.

3.2.1 Ingredient 1: Generate documentation with cross-references. In the current development process in the studied setting, documentation is generated from the model using Rhapsody Publishing Engine (RPE). Generated documents are based on templates filled in with details from the model. Since the documentation is completely generated and not manually changed, it is by construction synchronized with the system model. In the OpenMBEE platform, VE allows engineers to create explicit cross-references between a view and a model, that are then stored in MMS. For example, a value in a table in a document edited in VE can be linked to a specific value in a system model. In our setting, we can try to avoid manually linking model to documentation, since we can benefit from the document generation process to include cross-references automatically.

Once these links are automatically established, we can then utilize the existing development process for the manual creation of traceability links between code and documentation. For example, the developer would link the method declaration for an operation in the state machine to the corresponding list item in the documentation. In this way, model, documentation, and code are all kept consistent with respect to certain structural elements. Clearly, this is not enough to claim complete consistency between the three views, but the existence of explicit correspondence links provides the possibility to implement simple consistency checks, as we will see in Ingredient 3. First, to enable linking the code to the documentation in practice, we propose Ingredient 2.

3.2.2 Ingredient 2: Allowing cross-references between code and model. In addition to the generated links between model and documentation, additional links are required between code and documentation. For this functionality, we are inspired by VE, which allows the explicit creation of cross-references (links) between views. An example of its usage is the linking of a number in the document to a value property of a block in the system model. We observe that this linking could also be used to create a correspondence relation between variables or operations in the model and their code counterparts. For example, an operation defined in a full port in the system model can be mapped to a method definition in the code. To do so, we propose to implement the core VE functionality as a plug-in for an IDE, thereby allowing the linking to happen immediately adjacent to the implementation. The idea is that the created links are then stored in MMS, in the same way that the links between the system model and the documentation are stored in MMS.

In practice, our proposal would be a plug-in for the CLion IDE for C/C++ development[3] that allows for the creation of cross-references separate from typing the values, but akin to editing the property of an object. Upon selection of a code element such as a method or

[2]https://www.nomagic.com/products/cameo-systems-modeler

[3]https://www.jetbrains.com/clion/

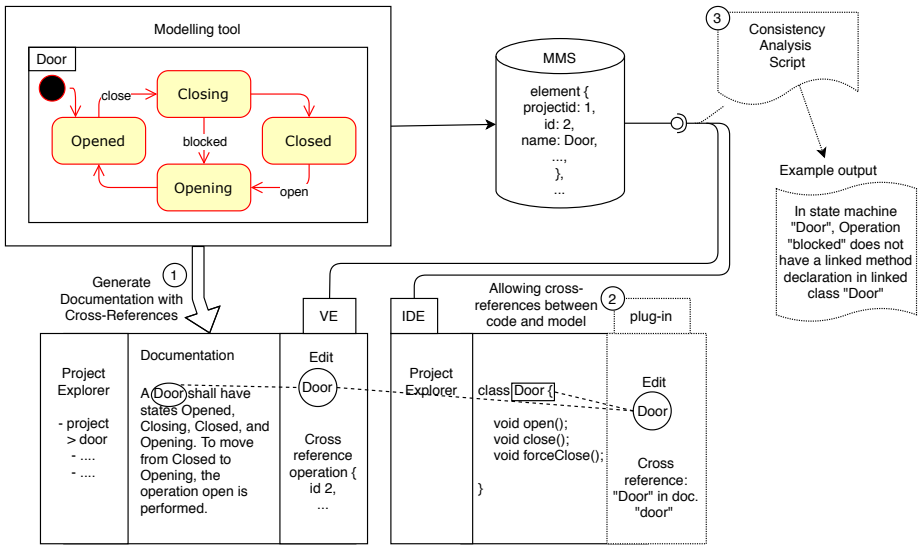


Figure 3: Schematic view of the proposed ingredients of the consistency checking approach in the OpenMBEE platform.

variable declaration, the plug-in should, in the separate pane, allow for the creation of a link between the selected abstract syntax tree node and a term from the documentation. Provided with this plug-in, the development process in the studied setting can be followed with minimal alterations. Indeed, documentation is generated now with links (ingredient 1) and the code is always written based on the documentation anyway. So, the resulting workflow would be that the code will be written based on the documentation as usual. Moreover, the software engineer manually links those code elements that can be explicitly linked to documentation elements.

Having explicit links between e.g. variable declarations and model elements allows for rudimentary structural consistency checks. To allow for more elaborate consistency checks, type information of model elements should be included in the *additionalProp* field of elements[4]. For example, we could go through all elements of type Full Port in MMS, get all operations defined within them, and check if all of those are linked to method definitions in the code.

3.2.3 Ingredient 3: Consistency analysis script. The third ingredient is proposed to utilize the newly created links between code, documentation, and model. Despite these links, consistency is not guaranteed by construction. Indeed, since linked elements can be syntactically synchronized using the MMS and MDK functionality, semantics is still not entailed. Even when model and code are consistent when the links are first created, the model may later be changed, leaving the already implemented code to be potentially inconsistent with the model. For example, a new child element may be added to the model element, which is not automatically reflected in the code. Therefore, we propose to introduce a consistency checking script that can detect inconsistencies based on the elements and references stored in MMS.

In general, consistency feedback should be provided frequently for it to be useful. However, constant updates during development may not be helpful, since many of the indicated inconsistencies would be trivial and a result of work in progress. As a middle ground, we propose to run the script as part of the continuous integration process for code development. In addition, the script should be executable by developers manually, on demand.

The script shall utilize the MMS API, and potentially the type information as proposed to be included in the system model. Our outlined structural consistency checks from Section 2 can then be implemented as queries on the MMS. For example, we can obtain all operations of state machines and check that they have a reference to a code element. If no such reference exists, we would indicate a potential inconsistency. Furthermore, we can utilize syntactical similarity scores, e.g. Levenshtein distance, to compare names of elements and indicate possible inconsistencies in cases where this score is lower than a predetermined threshold.

3.2.4 Limitations. The proposed setup is limited to structural consistency checking. No functional nor behavioral analysis of code is performed and we do not compare the functionality described in e.g. state machines and activity diagrams to the functionality of the code. In the studied setting, this is an acceptable limitation for two main reasons. Firstly, the model is mostly about documenting the structure, it focuses on dividing the system into components and describing how they are combined to form the system. The behavior is also included, but much less detailed. Indeed, it would be impossible to generate code directly from the model. Similarly, it is impossible to compare the high-level functional descriptions of activity diagrams to code. The second reason is that, given that structure is defined in more detail than behavior, structural discrepancies are expected to be a good indicator of potential errors in the implementation or model.

Another limitation is that the correspondence links between code and documentation elements have to be created manually. We see this is an acceptable limitation since the implementation

[4]https://mms.openmbee.org/alfresco/mms/swagger-ui/index.html

is done manually and it is based on the documentation already. Also, for new projects, there is not a large existing model yet, so correspondence links can be built up alongside the introduction of new features in design and implementation. Nevertheless, a future enhancement of the proposed approach could be to automatically discover correspondence links, which would be beneficial for application in existing projects with large pre-existing models and code bases. To achieve this, we anticipate mining the revision history for co-change information on a high level of granularity (e.g. files) [7]. To identify links at lower levels of granularity, we envision to utilize the names and types of elements. Through the well-defined development method in the described setting, we know between which types of elements traceability links are expected. Furthermore, the names of elements between which a link should be established are expected to be similar. Given this type and naming information, we might step-wise achieve an increasingly accurate prediction of traceability links [6].

4 DISCUSSION

We could also consider the adoption of SysMLv2 in the described setting. Both SysMLv2 with its API and MMS can play the role of an authoritative source of truth in an MBSE setting. Compliance of OpenMBEE with the SysMLv2 API is on the roadmap, but in a prototyping phase[5]. To move from the current industrial setting described in Section 2 to the proposed approach in Section 3 could require a considerable effort, e.g. adopting a new modeling tool or implementing extensions to the OpenMBEE components. Therefore, possible alternative ways of obtaining the described model-code consistency checking should be explored. One possibility could be to adopt SysMLv2 and utilize its APIs to provide a similar consistency checking script as described in Ingredient 3 in Section 3. The challenge then would be to create the explicit correspondence links between model and code elements.

Indeed, we do not fully utilize the ability of the OpenMBEE platform to synchronize across various modeling tools in our setting, since the only model of interest is the system model. Nevertheless, the main potential strength of introducing the OpenMBEE platform in the studied setting is the automatic synchronization of model, documentation, and code, and the insights into their consistency that this may provide. The main weakness of our proposal is that it remains limited to structural consistency checks.

5 RELATED WORK

Conceptually, the presented approach fits the theoretical framework of consistency checking in which views are frequently synchronized. Perhaps the closest fit is single-underlying model (SUM) where they propose a single source-of-truth representation and then different views on it [1]. In a way, MMS provides that here, the model is a single source-of-truth and then both documentation and code are views on it. Although we have an authoritative source of truth, it is not quite the same as a SUM, since the views are only on synchronized at the points where they have the explicit cross-references. In the SUM approach, all views are a view of the single underlying model and any change to any view is in fact a change to the SUM. MMS provides APIs to synchronize views in e.g. view editor with the model.

There are several related reports on experiences of adoption OpenMBEE. Although this paper is not an experience report, we can draw inspiration from the best practices as outlined by others such as [10]. Best practices of executable systems engineering (ESEM) have been gathered in the OpenSE cookbook [8], but in our studied setting, the model is not aimed to be executable, so we are more interested in comparing the model to the code.

6 CONCLUSION

In this paper, we share our ideas for including code as a view in the OpenMBEE framework to enable model-code gap analysis through structural consistency checks. We have presented our ideas of how this might work and look forward to discussing further possibilities with the community. Areas deserving special attention are versioning and variability of the model. In future work, we aim to at least partially implement our ideas in an industrial setting. We think the studied setting can benefit even from adopting OpenMBEE without our suggested additional ingredients, but it would be interesting to see if the suggested additions could provide much needed automated support for checking the consistency between the system model and the implementation.

ACKNOWLEDGMENTS

This work is supported by Software Center[6].

REFERENCES

[1] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. 2010. Orthographic Software Modeling: A Practical Approach to View-Based Development. In *Evaluation of Novel Approaches to Software Engineering*. Springer Berlin Heidelberg, 206–219. https://doi.org/10.1007/978-3-642-14819-4_15

[2] Robert Balzer. 1991. Tolerating inconsistency. In *Proceedings of the 13th international conference on Software engineering*. IEEE Computer Society Press, 158–165.

[3] Christopher Delp, Doris Lam, Elyse Fosse, and Cin-Young Lee. 2013. Model based document and report generation for systems engineering. In *2013 IEEE Aerospace Conference*. IEEE, 1–11.

[4] Sanford Friedenthal, Alan Moore, and Rick Steiner. 2014. *A practical guide to SysML: the systems modeling language.* Morgan Kaufmann.

[5] ISO/IEC/IEEE. 2011. *ISO/IEC/IEEE 42010:2011(E) Systems and software engineering – Architecture description.* Technical Report. 1–46 pages. https://doi.org/10.1109/IEEESTD.2011.6129467

[6] Robbert Jongeling, Johan Fredriksson, Federico Ciccozzi, Antonio Cicchetti, and Jan Carlson. 2020. Towards Consistency Checking Between a System Model and its Implementation. In *International Conference on Systems Modelling and Management ICSMM, 25 Jun 2020, Bergen, Norway.*

[7] Huzefa Kagdi, Jonathan I Maletic, and Bonita Sharif. 2007. Mining software repositories for traceability links. In *15th IEEE International Conference on Program Comprehension (ICPC'07)*. IEEE, 145–154. https://doi.org/10.1109/ICPC.2007.28

[8] Robert Karban, Amanda G Crawford, Gelys Trancho, Michele Zamparelli, Sebastian Herzig, Ivan Gomes, Marie Piette, and Eric Brower. 2018. The OpenSE Cookbook: a practical, recipe based collection of patterns, procedures, and best practices for executable systems engineering for the Thirty Meter Telescope. In *Modeling, Systems Engineering, and Project Management for Astronomy VIII*, Vol. 10705. International Society for Optics and Photonics, 107050W.

[9] Rainer Koschke and Daniel Simon. 2003. Hierarchical Reflexion Models.. In *WCRE*, Vol. 3. 186–208.

[10] Benjamin Kruse and Mark Blackburn. 2019. Collaborating with OpenMBEE as an Authoritative Source of Truth Environment. *Procedia Computer Science* 153 (2019), 277–284.

[11] David D Walden, Garry J Roedler, Kevin Forsberg, R Douglas Hamelin, and Thomas M Shortell. 2015. *Systems engineering handbook: A guide for system life cycle processes and activities.* John Wiley & Sons.

[5] https://www.openmbee.org/roadmap.html

[6] www.software-center.se