Integrating semantically-related Legacy Models in Vitruvius

Manar Mazkatli
Karlsruhe Institute of Technology
Karlsruhe, Germany
manar.mazkatli@kit.edu

Erik Burger
Karlsruhe Institute of Technology
Karlsruhe, Germany
burger@kit.edu

Jochen Quante
Robert Bosch GmbH
Renningen, Germany
Jochen.Quante@de.bosch.com

Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolek@kit.edu

ABSTRACT

The development of software-intensive systems, such as automotive systems, is becoming more and more complex as several modeling formalisms and languages are used to describe the same system from different viewpoints. These heterogeneous models can share common semantics and are usually separately developed and reused in different projects. This poses a challenge to the developer to keep them consistent along the development process. The VITRUVIUS approach for view-based software development provides change-driven consistency preservation between heterogeneous models. VITRUVIUS uses predefined consistency rules to support the consistent development of heterogeneous models from scratch. However, VITRUVIUS requires that models are created from scratch in order to track correspondence links between model elements. It does not support importing more than one existing model into its consistency preservation mechanism.

This paper extends VITRUVIUS with semi-automated *legacy model integration*, i.e. the ability to import multiple existing models into the consistency preservation mechanism. For this purpose, we propose an algorithm for automatic consistency checking of multiple existing models and for semi-automatic resolving of the potential conflicts. This algorithm is evaluated by a case study from automotive systems development. In this case study, we integrate existing models in the languages SysML, AMALTHEA and ASCET.

KEYWORDS

Consistency preservation · legacy models · ensure the consistency · conflicts resolution

ACM Reference Format:
Manar Mazkatli, Erik Burger, Jochen Quante, and Anne Koziolek. 2018. Integrating semantically-related Legacy Models in Vitruvius. In *Proceedings of Modelling in Software Engineering Co-located with ICSE (MiSE 2018).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

MiSE 2018, May 2018, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Model-based development becomes more and more popular for complex systems. For such systems, developers use different domain-specific modelling notation to describe the same system from different perspectives. The resulting heterogeneous models share semantics but are separately developed. This requires keeping them consistent over time, which is usually done manually.

In the automotive domain, for example, the standards SysML and AUTOSAR are widely used in conjunction with specific platforms such as AMALTHEA or ASCET. Often, the developers keep these models consistent manually by exchanging files and by importing changes. They reuse the information using also error-prone techniques, e.g., copy and paste. This can lead to an inconsistent description of the system. The potential inconsistency can affect the program code generated from these models negatively. Correction of the inconsistency errors at the assembly stage is very time-consuming due to the long compilation time. Furthermore, resolving the inconsistency only in the code level leads to drift and erosion between the code and model. This leads to inaccurate simulations and analyses, for example security or performance analyses.

The model-driven VITRUVIUS approach [10] offers a declarative definition of consistency between heterogeneous modelling artefacts and automate the consistency preservation process. In our prior work, we show how the VITRUVIUS can be used to support the consistent development of new automotive systems using the consistency mechanisms from the start [17]. This development scenario keeps the consistency between the models that are developed from scratch. To use VITRUVIUS in existing development scenarios, with existing *legacy* models, this require integrating them in VITRUVIUS development process by defining the corresponding elements and ensuring the consistency after the integration. Currently, there is a method to integrate single models into a VITRUVIUS repository [14]. In this paper, we present the legacy models integration process that makes the VITRUVIUS approach applicable for scenarios where multiple models with unspecified consistency relations exist. This process (1) supports using VITRUVIUS in the case of reusing the legacy models in the development process and (2) benefits of VITRUVIUS to check and reinstantiate the consistency in the case of using modelling tools independent of VITRUVIUS prototype.

We use a case study that describes the development of an onboard control unit for automotive systems, using the languages and standards SysML, AMALTHEA and ASCET. We have reused the metamodels as well as the consistency rules between them, which are defined in our prior work [17]. Moreover, we defined an import procedure for existing models and mechanism for checking the consistency between them and resolving the potential conflicts.

After a description of the foundations of the VITRUVIUS approach in Section 2, we will introduce our case study of an automotive software controller in Section 3. The LMI process is explained in Section 4. In Section 5, we describe the application of VITRUVIUS in our case study. Sections 6 and 7 contain related work and the conclusion.

2 FOUNDATIONS: VITRUVIUS

VITRUVIUS [11] is a view-based, model-driven framework for the management of heterogeneous models, i.e., models that are instances of different metamodels. This section is taken from [17]. VITRUVIUS is based on the concept of a *single underlying model (SUM)* [1], which represents all the information that is available about the system under development, and implements this concept into a *virtual SUM (VSUM)* that encapsulates models and enriches them with correspondence information and specialized views. The VSUM conforms to a customized metamodel that is specific to the domain in which the VITRUVIUS approach is used; for example, in the automotive domain, it may contain the metamodels of SysML, AMALTHEA and other standards, which are combined to form a modular SUM metamodel (see Figure 1). The metamodels are included non-intrusively and do not have to be adapted. To express the semantic relations between the elements of the metamodels, VITRUVIUS defines a language framework for consistency description and restoration that consists of three languages for *reactions*, *mappings* and *invariants*. Since VITRUVIUS is a view-based approach, all information in the SUM can only be retrieved or manipulated via specialized views. The consistency preservation mechanism is triggered by changes to one or several views. The preservation mechanism of VITRUVIUS then reacts on a list of changes to propagate them to the SUM.

VITRUVIUS has been implemented as a prototype[1], in the Eclipse Modeling Framework and can thus be used with any Ecore-conforming metamodel. So far, it has been applied to software architecture models [12] and model-based representations of programming languages [13]. Outside of pure software engineering, it has been applied in the systems modeling of energy networks [5].

Leonhardt et al. [14] have introduced two strategies to integrate one legacy model into VITRUVIUS. The first one simulates the recreation of the legacy model, whereas the second one is based on model generating tools (e.g., model-to-model transformations) to generate the corresponding model of the legacy model and link them in the change-based development environment. However, these strategies integrate only one legacy models in change-driven development, whereas our concept allows to link and integrate two or more semantically-related legacy models in VITRUVIUS platform.

[1] https://sdqweb.ipd.kit.edu/wiki/Vitruvius

3 CASE STUDY: ENGINE CONTROLLER

This section describes a case study from automotive system domain at Robert Bosch GmbH Corporate Research, which has been also introduced in our prior work [17]. This work ensures the consistency between the existing models of this case study and integrates them in VITRUVIUS consistent system development process.

The following sub-sections give an overview of the pieces of software under development, the languages that are used to develop it, and the consistency problem that arises.

3.1 PID Controller Software

This case study describes the development of a controller software using a traditional PID (Proportional, Integral, and Derivative) based on Control Algorithm (CA). This algorithm is commonly used in the automotive field (e.g., for controlling throttle positions). It depends on a control loop feedback mechanism to calculate an error value as the difference between a desired set point (`target_position` in our case study) and a measured process variable (`actual_position` in our case study). Then it attempts to minimize this error over time by adjustment of a control variable (`new_position` in our case study) through applying proportional, integral, and derivative terms.

In this development scenario, developers describe the structure of the controller software in *SysML*. They use the *AMALTHEA* platform to describe the software architecture and to generate glue-code describing tasks code, operating system configuration, and the scheduling. To implement the defined software architecture, developers use *ASCET* and generate the implementation code automatically.

3.2 Languages and Models

This section presents the modelling tools *SysML*, *AMALTHEA* and *ASCET*, which are used to model PID case study.

3.2.1 SysML. The Systems Modelling Language (SysML) [18] is a graphical modelling language developed for systems engineering. It can depict a system's structure using Block Definition Diagrams (BDD) and Internal Block Diagrams (IBD). The BDDs provide a black box representation of a system's blocks as well as the interconnections between them, using the concept of *flow ports*. The IBDs instantiate the BDDs to represent the final assembly of all blocks within the main system block. For example, we describe the controller software structure of our case study using an IBD. Figure 2 shows the main Block (ControlAlgorithm) with its internal blocks (PID Tuning block and Limiter block) in addition to its in-/out-ports (`actual_position`, `target_position` and `new_position`).

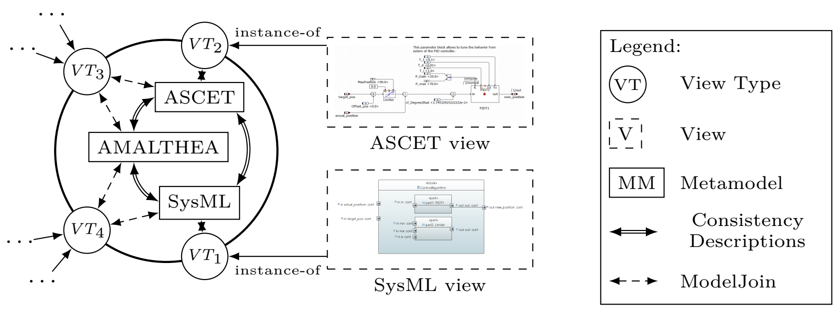

Figure 1: The modular SUM Metamodel concept of Vitruvius at the example of automotive systems engineering from [17]

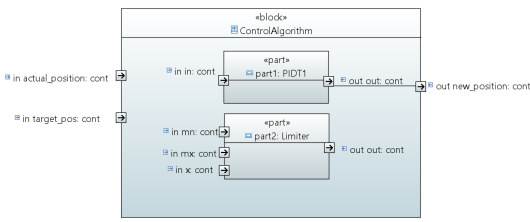

Figure 2: Modelling the structures of CA using IBD [17]

3.2.2 AMALTHEA. AMALTHEA is an open and expandable tool platform for embedded multicore systems. It combines tools that are used to develop multi-core automotive ECUs in a single platform [3, 7].

Both hardware and software can be modelled in AMALTHEA. Developers describe the software architecture using component and software models. Components models define the system components and the connections between. Software model defines units such as *Runnables* (executable software units that can run in parallel), *Labels* (data elements located in memory that can be read or written by runnables) and *Processes* (generalization of tasks and interrupt service routines defining execution paths that call runnables or other processes). In our case study, we have AMALTHEA models defining one software component called *ControlAlgorithm*, which contains one Task instance calling two runnables (`CA_normal` and `CA_out`) that read/ write three labels (`new_position`, `target_pos`, `actual_position`). After the modelling the developers generate the C code (glue-code) including the tasks and OSEK Implementation Language files[2] that describe OSEK real time systems (multitasking and communication configuration).

3.2.3 ASCET. The Advanced Simulation and Control Engineering Tool (ASCET) is a tool suite from ETAS GmbH for model-based development of embedded automotive software. ASCET Modeling and Design (ASCET-MD)[3] is a part of the ASCET product family and offers executable specification of ECU functions. Developers can describe and model the behaviour of ECUs using graphical tools, such as block diagrams and state machines, or with textual tools such as Embedded Software Description Language (ESDL) editors, and C code editors. The block diagram editor describes the functionality of the components through blocks and shows the flow of the data or control signal between them. Moreover, it enables representing different data type and arithmetical/-logical operations.

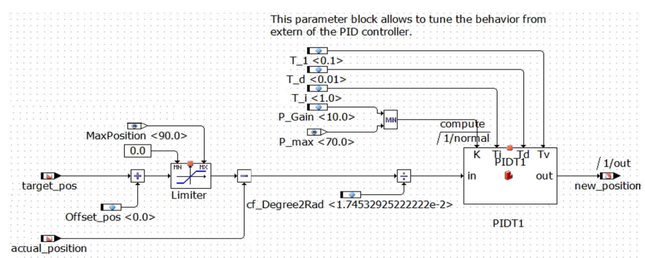

Figure 3: Modelling CA using ASCET block diagram editor [17]

For instance, Figure 3 shows the implementation of the ControlAlgorithm software component using the ASCET type AscetModule. The values of the input will be imported from other blocks in the form of input messages (`target_pos` and `actual_position`). Similarly, the output (`new_position`) will be exported to other components as an output message. Besides, the developers use two pre-defined components: the *Limiter* component provided from the ASCET library and the *PIDT1* component, whose functionality is described in a separate block diagram. After modelling the functionality of the component, developers can generate C code, which will be integrated with glue-code generated by AMALTHEA.

3.3 Consistency Preservation

Consistency preservation between the models that are separately developed is a challenge faced by automotive developers. The reason is because the synchronisation between most of these models can be classified according to [8] as

[2]seehttp://www.irisa.fr/alf/downloads/puaut/TPNXT/images/oil25.pdf, retrieved 2016-12-13
[3]see http://www.etas.com/de/products/ascet_md_modeling_design-details.php, retrieved 2016-12-13

full round-tripping synchronisation. According to their three-dimensional taxonomy, (1) the automotive models are organizational symmetry: there is no dominated model by the synchronisation, (2) they are informational symmetry: there are shared semantics between them in addition to the additional private details and (3) they can be incrementally synchronised, as Diskin et al. emphasize, considering both of organizational and informational symmetry. Moreover, most the automotive models generate their own code. According to [8] we can classify this generation as partial code generation that need more implementation details to be executed. An example of our case study, the code ($C_{AMALTHEA}$) generated from AMALTHEA model is supposed to be manually integrated with implementation details like method bodies, which is generated from the ASCET model (C_{ASCET}). Keeping the high level automotive models (e.g. AMALTHEA and ASCET models) consistent will allow generating an executable correct final code. The compilation of the final code can take hours. Therefore, it is an expensive process to correct the errors resulting by potential inconsistency at this stage. Hence, the goal of the developers is to ensure the consistency at the modelling level. To achieve this goal Bosch developers, for example, apply technology based on Manufacturer Supplier Relationship (MSR)[4] [23] and XML [25]. In this technology, the documents store needed information in a uniform format in a shared database to reduce the redundancy of information.

As explained in Section 1 such an approach suffers oft from the manual production, synchronisation and reuse of the information, which may lead to inconsistency. Besides, there is no efficient tool, as far as we know, to check and reinstantiate the consistency before generating and integrating the code.

4 LEGACY MODELS INTEGRATION

The VITRUVIUS approach supports consistent view-based development of heterogeneous models that are built from scratch, but does currently not support the integration of more than one legacy model. Such integration is also required in automotive system where there is high amount of re-use. Moreover, integrating the legacy models in VITRUVIUS enables ensuring the consistency between them even though they are developed and maintained separately using development tools that are not supported by VITRUVIUS.

Therefore, we introduce in the following section a process to integrate two or more existing models into the VITRUVIUS platform. Subsection 4.2 describes how to use this process to ensure consistency in multi-models system development.

4.1 Legacy Models Integration Process

This section describes the legacy models integration process which is shown in Figure 4. The process consists of three main steps. The first step initializes the VITRUVIUS platform manually, whereas the second and third steps describe the *Legacy Models Integrator (LMI)* algorithm, which checks the consistency between the correspondent models automatically, links the corresponding elements and resolves the potential inconsistencies semi-automatically. In the first step, the methodologists create VSUM metamodel containing the heterogeneous metamodels as well as the potential dependencies between them as described in [6]. While defining the correspondence rules (i.e. the dependencies between the metamodels), the methodologists have also to mark the *correspondence identifier* consisting of attributes/ references and conditions, which can identify whether a pair of artefacts are corresponding with each other or not. For instance, the *name* attribute is the identifier for the correspondence between a Component class of AMALTHEA and an AscetModule class of ASCET that describes the behaviour of the AMALTHEA Component. The resulting VSUM metamodel can be reused for different projects based on the same metamodels. After creating VSUM metamodel based on the defined correspondence rules, the methodologists will define the view types as well as the views.

The second step of the integration process is checking the consistency between the legacy models. To do so, LMI first traverses the legacy models sequentially using a traverse strategy similar to the strategy of Leonhardt et al. [14]. Then LMI links the related artefacts that comply with the defined corresponding rules including the defined correspondence identifier, if they have not been linked in previous traverse. Linking the related artefacts is necessary for checking and keeping the consistency between their references on one hand and for the further development of the models based on VITRUVIUS platform on the other hand. For linking the related objects LMI checks the similarity of the correspondence identifier For example, LMI links each an AMALTHEA Component object with an AscetModule object that has same name as the Component object. During traversing the models, LMI also lists the potential inconsistencies as well as the appropriate EMF changes that can resolve them based on VITRUVIUS synchronisation. LMI distinguishes between the following two types of inconsistency:

- The first type is the absence of any corresponding artefacts. An example of our case study is when the developers of ASCET define no AscetModule instance that can define the behaviour of an existing AMALTHEA Component object. To resolve this inconsistency LMI supposes that the existing element, whose correspondent elements do not exist, (AMALTHEA Component instance in our example) has just been created and generates an EMF create-change for it. The reason of this assumption is that VITRUVIUS synchronisation mechanism propagate such this EMF-create change to the related views by creating the missing correspondent elements automatically based on the defined correspondence rules. Hence, LMI saves the generated EMF-create change in a changes list, which will be synchronised by VITRUVIUS in the next step and resolves the inconsistency. In our example, generating an EMF-create change for an AMALTHEA Component element will create, after the synchronisation, an AscetModule instance named with the name of the AMALTHEA

[4] http://www.msr-wg.de, retrieved 2016-12-13

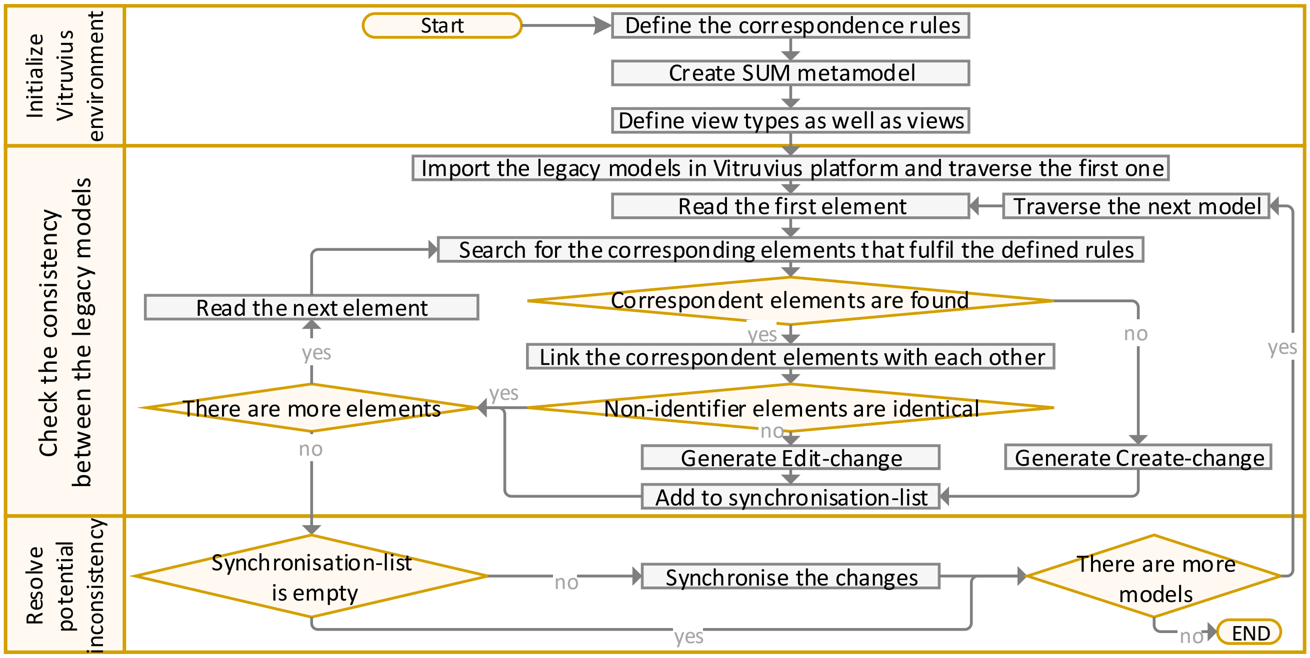

Figure 4: The legacy models integration process.

Component and link them which each other. Additionally, the synchronisation will create its references corresponding with the AMALTHEA Component references, supply them with the shared information and similarly link them with their corresponding elements.

- The second type of the inconsistency is a conflict between the values of the non-identifier related attributes. To resolve this conflict, LMI supposes that the value of one of these attribute has just been updated and generates the appropriate EMF edit-change according to this assumption. Similarly to the first type of inconsistency, the EMF edit-change will be saved and synchronised in the next step. Solving this type of conflict is based on the actions defined by VITRUVIUS Reactions language. These actions can resolve the conflict automatically (adopting a value of one of the attribute and updating accordingly the other one) or semi-automatically (asking the developer to determine the correct value and update the wrong one accordingly).

In the last step, LMI will resolve the potential inconsistencies. The changes listed in the previous step will trigger VITRUVIUS synchronisation, which in turn will propagate them to the related views. After that the second and third steps of the integration process will be repeated to traverse each of the remaining models.

4.2 Vitruvius development process using LMI

This section describes the development process based on VITRUVIUS platform. The development process begins with initializing the platform. Then, developers integrate their legacy models by LMI if they are already existing. Otherwise, they implement their new models. If the modelling tool is supported by the VITRUVIUS platform, then it will record/import the changes and propagate them automatically. Otherwise, the developer will import the models into VITRUVIUS from time to time to check the consistency and resolve the possible conflicts by LMI. Then they review the potential automatic changes and export the resulting consistent models to the modelling tools, either for further development or for generating the final code.

5 EVALUATION

To evaluate the applicability and feasibility of the approach, we have implemented the LMI process and applied it on our case study (presented in Section 3) and on an another artificial case study that is based on the same metamodel, but contains more inconsistencies. The evaluation covers the following: Detecting the related elements and linking them with each other through first building the mappings on instance level, second finding the inconsistent cases automatically, third generating the appropriate changes that can resolve them, and finally resolving the inconsistency by propagating the changes based on VITRUVIUS synchronization.

To do the evaluation, we reused the VSUM metamodel that is defined in our previous work [17] and includes the metamodels of SysML, ASCET and AMALTHEA as well as the correspondence rules between them. Table 1 lists the most important correspondences with examples.

SysML	AMALTHEA	ASCET	Examples from the case study
Block	Component	AscetModule	`ControlAlgorithm`
FlowPort	Label	Message	`target_pos, actual_position, new_position`
—	Runnable	Method	`normal, out`
—	Task	Task, InitTask, ...	`_10MS`

Table 1: Main correspondences between SysML, AMALTHEA and ASCET from [17]

Then we have implemented and executed LMI algorithm to integrate the legacy models of our case study and of the artificial one. The evaluation results are:

- LMI detects and links only the correspondent elements that fully comply the correspondence rules, such as the `ControlAlgorithm` Component from AMALTHEA, and the `ControlAlgorithm` AscetModule in ASCET. The different naming conventions between AMALTHEA Runnable and ASCET Method instances prevented LMI from detecting and linking these correspondent instances, since the correspondence identifier is the name attribute.
- LMI detects all inconsistencies that violate the correspondence rules. Table 2 lists the inconsistencies detected in the CA case study as well as the automatic resolution applied by LMI. The different naming conventions between the Runnable and Method instances caused the first four inconsistencies, since the correspondence rule is violated, and the corresponding elements are not found by LMI. The last inconsistency is the conflict between the values of the `_10MS` AMALTHEA Task priority attribute and `_10MS` ASCET SoftwareTask priority attribute.
- LMI could generate the appropriate changes and propagate them correctly. For example, LMI resolved the first inconsistency by generating a create-change for the runnable `CA_out`. Synchronising this change created the missing Method instance named `CA_out` having neither arguments nor a return type and arranged it correctly to the methods list of the `ControlAlgorithm` element of type AscetModule. Similarly, LMI resolves the second, third and fourth inconsistency. This resolution would be correct if the corresponding elements were really not existing. But in this case (different naming conventions), LMI will recreate the existing correspondent elements with other names, e.g., LMI created additional Runnable instances (`normal, out`) and Method instances (`CA_normal, CA_out`).
- LMI could not resolve the inconsistencies fully automatically. In the last case, LMI asked the developer about the correct value of the priority attributes, because the methodologist had not chosen one of these correspondent elements (AMALTHEA Task and ASCET SoftwareTask) as a dominant element, which of the attributes' values are chosen in conflict cases.
- LMI decreases the time needed to find the inconsistencies compared to the traditional approach, where generating, compiling and debugging the code may take hours to detect and fix the inconsistency.
- Developers have rolled back the changes applied by LMI to resolve the difference of naming convention, correct the names and ensure the consistency.
- VSUM has been unproblematically reused to integrate other artificial models based on the same metamodels.

In summary, we can say that first, the case study shows that LMI can detect and link the related element, as long as there is no difference in naming conventions. Second, the case study confirms the automatic checks of the consistency, since LMI could detect the inconsistencies that cause compile/runtime errors. Third, the evaluation validates the Vitruvius synchronization mechanism since the changes generated by LMI are propagated correctly to the related elements. Fourth, the evaluation lists some cases where resolving the potential inconsistency cannot be done fully automatically, e.g., resolving the conflicts between the non-identifier attributes. Another example is resolving the conflicts in the case of a one-to-many mapping, such as the correspondence rule defined for the relation between AMALTHEA tasks and ASCET tasks. In this case, if LMI should create an ASCET task instance, it will ask the developer to choose one of the following ASCET tasks: Task, InterruptTask, InitTask, SoftwareTask, TimeTableTask. Finally, the case study highlighted the importance of reviewing the performed changes to verify the automatic resolution, roll back the undesirable changes, such as the changes made to resolve the naming, or supply the unshared information, such as defining the execution paths for the AMALTHEA Task created by Vitruvius.

6 RELATED WORK

In automotive field, there are various approaches to ensure the consistency between the heterogeneous models developed separately. The old document-centric approach suffers (as mentioned in Section 1) from the manual preservation of consistency. Therefore, Born et al. [2] suggest representing the exchanged information (even if it is a document) as a model to ease tracing them and perform semi-automatic checks of consistency. Our approach allows automatic checks, semi-automatic conflict resolution.

Other works specify the relations between the models and use them to perform automatic consistency preservation,

Inconsistencies	Generated EMF changes	Change synchronization results
CA_out Method containing neither arguments nor return type is expected in ControlAlgorithm AscetModule.	Create-change for CA_out Runnable.	The missed CA_out Method is created.
CA_normal Method containing neither arguments nor return type is expected in ControlAlgorithm AscetModule.	Create-change for CA_normal Runnable	The missed CA_normal Method is created.
out Runnable is expected in ControlAlgorithm Component.	Create-change for out Method	The missed out Runnable is created.
a Runnable instance named normal is expected in ControlAlgorithm Component.	Create-change for normal Method	The missed normal Runnable is created.
a conflict between the values of AMALTHEA Task Priority and ASCET SoftwareTask Priority attributes.	EMF Edit-change for AMALTHEA Task	priority values are updated according to developer choice.

Table 2: Inconsistencies found in the CA case study

like the work of Giese et al., who use Triple Graph Grammars (TGGs)to describe the relationship between SysML and AUTOSAR [9]. TGGs define bidirectional model-based transformations to synchronize these two models automatically or generate one model from its corresponding model. In comparison to Giese approach, Vitruvius approach offers the consistent development and management of the different heterogeneous models and the contribution of this paper allows also linking and integrating more than one legacy model, which is demand in large system development.

Salay et al. [19] apply a concept similar to Vitruvius to keep consistency in vehicle control systems based on *macromodels*. Macromodels define mappings between models and constraints using formal methods, detect inconsistencies, and may repair them using formal expressions of model relationships. Compared to our approach, Salay et al. approach does not support the case of legacy models.

Other concepts keep consistency in automotive development by generating a consistent model from another existing one using model-based transformations. For example, Selim et al. [20] apply model transformations to migrate from legacy domain-specific models of General Motors to standardized AUTOSAR models. Model-to-model transformations have been also used by Sindico et al. [21] in order to generate Simulink models from SysML models and vice versa. Similar work by Sjöstedt et al. [22] transforms Simulink models to UML composite structure and activity models based on Atlas Transformation Language ATL. The approaches mentioned in this paragraph are limited to a specific combination of two models or languages and do not allow the further consistent development of the source model and the generated one.

Macher et al. [15] depend on the seamless combination of heterogeneous tools approach [4] to exchange the models between SysML and Matlab/Simulink tools. Other work of Macher [16] based on the same approach generates the configuration of RTOS from control system information in SysML and vice versa. Macher's approach is limited also to the used tools and cannot ensure the consistency between two legacy models. Vierhauser et al. [24] present a framework for checking and maintaining consistency between the Product Line (PL) model and some parts of underlying code. However, their approach is limited to PL models and their underlying code base and does not support other models needed in software product line engineering.

7 CONCLUSION

In this paper, we have presented the legacy models integration process that makes Vitruvius applicable in scenarios with existing models of different languages.

On one hand, this work will support the reuse of the legacy models and enable developing them in Vitruvius platform, which automates steps of consistency preservation using change-based consistency rules. This will allow benefiting from Vitruvius synchronisation process that propagates the changes to the related artefacts and use predefined action to resolve the potential conflicts.

On another hand, our work allows ensuring the consistency between the heterogeneous models that are developed using external modelling tools. It will provide automatically checking the consistency. Moreover, it applies predefined actions to resolve the potential conflicts semi-automatically. Above all, it avoids the manual overhead needed to keep the consistency and increase the reliability of the system under development. In a future work, we aim to avoid recreating the existing artefacts that are not detected by LMI due to literal errors. To achieve that, we aim to nominate the artefacts, which have a great similarity to the expected artefacts, to be checked by the developer before applying the predefined reaction.

REFERENCES

[1] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. "Orthographic Software Modeling: A Practical Approach to View-Based Development". In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by LeszekA. Maciaszek, César González-Pérez, and Stefan Jablonski. Vol. 69. Communications in Computer and Information Science. Berlin/Heidelberg: Springer, 2010, pp. 206–219.

[2] Marc Born, John Favaro, and Olaf Kath. "Application of ISO DIS 26262 in practice". In: *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness and Safety*. ACM. 2010, pp. 3–6.

[3] Christopher Brink and Jan Jatzkowski. *AMALTHEA (ITEA2 – 09013) – White Paper*. Tech. rep. University of Paderborn, Germany, 2013.

[4] Manfred Broy et al. "Seamless model-based development: From isolated tools to integrated model engineering environments". In: *Proceedings of the IEEE* (2010).

[5] Erik Burger, Victoria Mittelbach, and Anne Koziolek. "View-based and Model-driven Outage Management for the Smart Grid". In: *Proceedings of the 11th Workshop on Models@run.time co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016)*. (Saint Malo, France). CEUR Workshop Proceedings, Oct. 2016.

[6] Erik Burger et al. "View-Based Model-Driven Software Development with ModelJoin". In: *Software & Systems Modeling* 15.2 (2014). Ed. by Robert France and Bernhard Rumpe, pp. 472–496.

[7] The Amalthea consortium. *AMALTHEA ITEA 2 - 09013, Deliverable D1.1 State of the art of Design Flow and verification methods and tools*. Tech. rep. ITEA 2, 2011.

[8] Zinovy Diskin et al. "Towards a Rational Taxonomy for Increasingly Symmetric Model Synchronization". In: *Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings*. 2014, pp. 57–73.

[9] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. "Model synchronization at work: keeping SysML and AUTOSAR models consistent". In: *Graph transformations and model-driven engineering*. Ed. by Gregor Engels et al. Springer, 2010.

[10] Max E. Kramer, Erik Burger, and Michael Langhammer. "View-centric engineering with synchronized heterogeneous models". In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. New York, NY, USA: ACM, 2013.

[11] Max E. Kramer, Erik Burger, and Michael Langhammer. "View-centric engineering with synchronized heterogeneous models". In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. VAO '13. Montpellier, France: ACM, 2013, 5:1–5:6.

[12] Max E. Kramer et al. *Realizing Change-Driven Consistency for Component Code, Architectural Models, and Contracts in Vitruvius*. Tech. rep. Karlsruhe: Karlsruhe Institute of Technology, Department of Informatics, 2015.

[13] Michael Langhammer and Klaus Krogmann. "A Coevolution Approach for Source Code and Component-based Architecture Models". In: *17. Workshop Software-Reengineering und-Evolution*. Vol. 4. 2015.

[14] Sven Leonhardt et al. "Integration of Existing Software Artifacts into a View- and Change-Driven Development Approach". In: *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*. MORSE/VAO '15. L'Aquila, Italy: ACM, 2015, pp. 17–24.

[15] Georg Macher, Eric Armengaud, and Christian Kreiner. "Integration of Heterogeneous Tools to a Seamless Automotive Toolchain". In: *Systems, Software and Services Process Improvement*. Ed. by Rory V. O'Connor et al. Springer International Publishing, 2015.

[16] Georg Macher et al. "Automotive real-time operating systems: a model-based configuration approach". In: *ACM SIGBED Review* (2015).

[17] Manar Mazkatli et al. "Automotive Systems Modelling with Vitruvius". In: *15. Workshop Automotive Software Engineering*. (Chemnitz). Vol. P-275. Lecture Notes in Informatics (LNI). Gesellschaft für Informatik, Bonn, 2017, pp. 1487–1498.

[18] *OMG Systems Modeling Language (OMG SysML)*. Version 1.3. Object Management Group. June 2012.

[19] Rick Salay, Shige Wang, and Vivien Suen. "Managing related models in vehicle control software development". In: *Model-Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012*. (Innsbruck, Austria). Ed. by Robert B. France et al. Berlin, Heidelberg: Springer, 2012, pp. 383–398.

[20] Gehan MK Selim et al. "Model transformations for migrating legacy deployment models in the automotive industry". In: *Software & Systems Modeling* 14.1 (2015), pp. 365–381.

[21] Andrea Sindico, Marco Di Natale, and Gianpiero Panci. "Integrating SysML with Simulink using Open-source Model Transformations." In: *SIMULTECH*. Ed. by Janusz Kacprzyk, Nuno Pina, and Joaquim Filipe. SciTePress, 2011, pp. 45–56.

[22] Carl-Johan Sjöstedt et al. "Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations". In: *OMER4 Workshop: 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems*. 2008, pp. 137–160.

[23] Marek Szwejczewski, Fred Lemke, and Keith Goffin. "Manufacturer-supplier relationships: An empirical study of German manufacturing companies". In: *International Journal of Operations & Production Management* 25.9 (2005), pp. 875–897.

[24] Michael Vierhauser et al. "Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2012, pp. 531–545.

[25] Bernhard Weichel and Martin Herrmann. *A backbone in automotive software development based on XML and ASAM/MSR*. Tech. rep. SAE Technical Paper, 2004.