# Towards agile collaborative multi-view modeling with inconsistency tolerance

Yaser Shahvari[1,†], Mohammadreza Sharbaf[1,*,†], Shekoufeh Kolahdouz Rahimi[1,2,†] and Sobhan Yassipour Tehrani[3,†]

[1] *MDSE Research Group, Department of Software Engineering, University of Isfahan, Isfahan, Iran*
[2] *School of Arts, University of Roehampton, London, U.K*
[3] *Computer Science Department, Faculty of Engineering, University College London, London, U.K*

**Abstract**
Model-driven engineering (MDE) copes with the complexity of software development by using the principles of separation of concerns and automatic transformation. In MDE, stakeholders from diverse domains collaborate concurrently on different models to quickly analyze, design, and generate complex software-intensive systems. To adopt an agile development approach in MDE, an inconsistency tolerance framework is needed. This framework postpones the resolution phase for incompatibilities caused by online cooperation, allowing temporary incompatibilities to be automatically fixed while requiring resolution only for the remaining conflicts at the appropriate time. In this paper, we propose a general framework for identifying and diagnosing inconsistencies, making decisions regarding inconsistency tolerance or intervention for resolution, and outlining tolerance strategies in agile systems modeled as multi-views. Our framework comprises three primary phases: detection, analysis, and tolerance. It is designed to identify inconsistencies in multi-view models, which are inherently more complex than single-view models. The tolerance phase employs strategies to tolerate inconsistencies, enhancing the flexibility of the agile development approach for collaborative multi-view modeling.

**Keywords**
Agile MDE, Collaborative Modeling, Inconsistency Tolerance, Multi-view Modeling

## 1. Introduction

During the development of complex and large software systems, developers use several languages and modeling tools and different views to describe a system. Multi-View based development is a suitable approach for dealing with complex systems and is also recognized in other engineering disciplines. However, we have to deal with consistency problems [1].

While many researches have been conducted on inconsistencies, most approaches focus on maintaining consistency in terms of syntactic relationships between models [2-6]. Instead of simply removing inconsistencies from the system, Finkelstein suggests, sometimes consistency should be managed. This requires the detection and identification of inconsistencies before

[*] Corresponding author.
[†] These authors contributed equally.

yaser_shahvari@eng.ui.ac.ir (Y. Shahvari); m.sharbaf@eng.ui.ac.ir (M. Sharbaf); shekoufeh.rahimi@roehampton.ac.uk (S. Kolahdouz Rahimi); sobhan.tehrani@ucl.ac.uk (S. Yassipour Tehrani)

their resolution and subsequent analysis. However, inconsistencies are state entities that may occur, evolve, and later potentially disappear as a natural consequence of a design workflow.

Agile software development is an iterative and gradual approach to software development based on a set of concepts and principles [7]. An Agile MDE engineering approach used to develop a process can integrate MDE development principles and tools into existing agile software processes or incorporate features of agile approaches into MDE [8].

In the rest of the paper, our motivating example will be described in Section 2. Then in Section 3, we present our proposed solution in the form of a framework to tolerate inconsistency. Finally in Section 4, we conclude the paper and discuss possible future works.

## 2. Motivating example

A large logistics company is developing an AGV control system to optimize warehouse operations. In the development cycle with the Agile method, different views have the following needs and features:

- Warehouse Managers' View: Monitoring AGV performance, managing routes.
- Operators' View: Manual control in emergency situations and initial configurations.
- Maintenance Team's View: Monitoring AGV status, planning maintenance and repairs.
- Data Analysis Team's View: Collecting and analyzing AGV performance data.

In the next step, each view is developed based on the mentioned needs and features. During the development of different views, many inconsistencies may arise.

For example, The operators' view requires a simple and quick interface for manual control of AGVs, while the maintenance team's view needs accurate and up-to-date data. Inconsistency tolerance allows the team to develop these different needs in parallel and manage any conflicts that arise. Inconsistency tolerance in the development of the AGV control system using Agile methodologies helps the development team continuously receive user feedback and manage existing differences and conflicts. By using quantification of semantic inconsistencies, microservices architecture, continuous prototyping and testing, and regular reviews, the team can identify and gradually resolve inconsistencies, ensuring the final system meets the diverse needs of users.

## 3. A framework for tolerating inconsistency

As mentioned earlier, inconsistency tolerance is very important in using agile techniques. When we do not use inconsistency tolerance for system development, many times when conflicts and inconsistencies occur, we have to stop the project to fix them, which is not in line with our expectation of agile techniques. Our proposed framework consists of three main phases, which include the identification phase, the analysis phase, and the tolerance phase. Figure 1 shows a view of this framework.

The difference between our approach and the previous works is that, firstly, the work done in each of these phases is considered for complex systems and in multi-view mode, which will be explained later. Second, we have considered a separate phase for tolerating inconsistency called

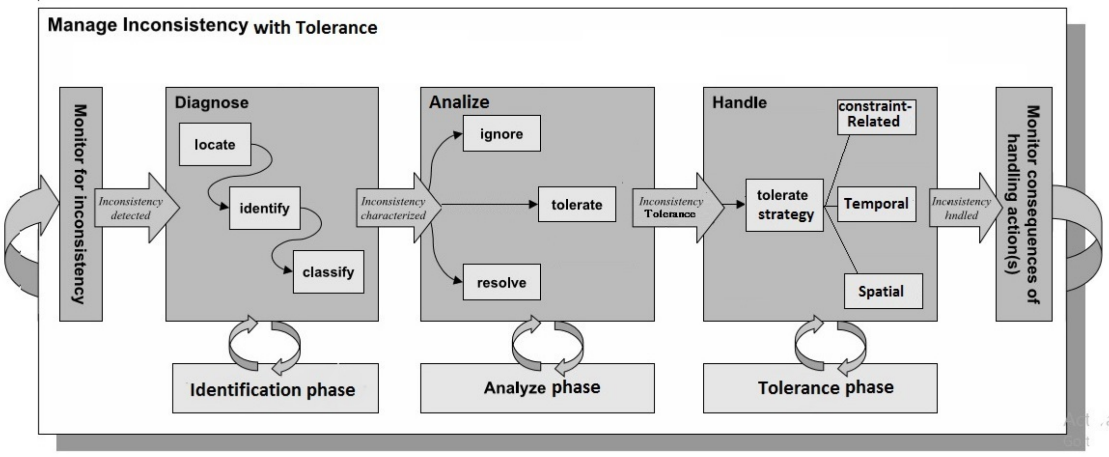

**Figure 1:** A framework for tolerable inconsistency management.

Tolerance Phase, and in this phase, we use an appropriate inconsistency tolerance strategy according to the inconsistencies identified in the previous stages.

## 3.1. Identification phase

Since in multi-view modeling we may deal with several engineering teams and different modeling tools, it is more difficult to identify inconsistencies than in single-view process modes. Basirati et al. [9] proposed a 5-step framework to identify and diagnose inconsistency, we will use the same model to detect inconsistencies in multi-view modeling.

After identifying inconsistencies in this phase, we classify them. Inconsistencies in multi-view modeling are divided into three main categories: syntactic, structural and semantic. Syntax inconsistencies occur when there are errors or inconsistencies in the format or syntactic structure of models. These types of inconsistencies are related to formatting and how to display information in models. Structural inconsistencies occur when there are conflicts or differences in the structure and order of components or relationships between them in different views. These types of inconsistencies are related to the way components are organized and related in the models. Semantic inconsistencies occur when there are contradictions or differences in the meanings and concepts of components or the relationships between them in different views. These types of inconsistencies are related to the concept and meaning of the information in the models.

## 3.2. Analysis phase

In the analysis phase, a decision is made regarding how the system functions in the face of inconsistencies. The choice of inconsistency management strategy depends on the context and the impact it has on other aspects of the development process. In the early stages of development, commonly known as the design and analysis stages, inconsistency tolerance may be used due to the need for speed in development or lack of access to sufficient information about the system. But in later stages such as implementation and testing, it may be necessary to resolve inconsistencies to prevent further problems and maintain system quality. Also, if the inconsistency has a major impact on the performance or quality of the system, its resolution is considered a top priority. But if the inconsistency has little impact and system development can continue, it may make more sense to tolerate it. On the other hand, in some fields of work, such as medical or space fields, solving inconsistencies is considered very critical and no major inconsistencies are tolerated. While in other contexts with the least possible side effects, tolerance of inconsistency may be permissible. Ultimately, the decision whether to tolerate or resolve inconsistency must be made according to the specific circumstances of each project, development team, user needs, and organizational goals.In some cases, trying to fix an inconsistency has undesired consequences. In such cases, developers may choose to ignore the inconsistency in their descriptions. Good practice dictates that such decisions should be reviewed as the project progresses or as the system evolves.

### 3.3. Tolerance phase

In this phase, according to the classification of the detected inconsistencies, inconsistency tolerance strategies are used. Tolerating inconsistency means managing and accepting differences and conflicts in such a way that these conflicts do not hinder the overall progress of the development process. To effectively manage and tolerate inconsistencies in multi-view modeling, specific strategies can be used for each type of inconsistency (syntactic, structural, and semantic). These strategies are well explained in reference [10].

General strategies include Constraint Related, Temporal and Spatial. In Constraint Related Inconsistency Tolerance, we use techniques such as ranking or weighting constraints and scoring solutions based on the number of satisfied weighted constraints. The process of prioritization and sorting is often referred to as ”relaxation”. In most collaborative multi-view projects, we recommend Temporal Inconsistency Tolerance. In most approaches, inconsistencies can be tolerated until consistency is restored, so that corrections are postponed until this point. For multi-site distributed developments, a variable threshold for inconsistencies is defined by a time interval in which more inconsistencies are accepted at the beginning and fewer at the end. Spatial approaches ensure that consistency is improved to a limited extent. Case-based retrieval ensures that any part of the model that was previously consistent is still consistent afterwards. For efficiency reasons, only a subset of ”relevant” cases can be determined, i.e., a scope of influence is computed for changes, and then checked as for case-based restorers. Measure-based restorers guarantee that a chosen measure of consistency is not reduced by the restoration process[10].

## 4. Conclusion

When we use agile approaches in software development, tolerance of inconsistency causes the development of the system to continue continuously and prevents the project from stopping. In this paper, a general framework for the management of inconsistencies was presented, which includes the diagnosis, analysis and tolerance of inconsistencies. Considering that the topic discussed in this paper is about agile systems that are modeled as multi-views, therefore, the process of detecting inconsistencies in this case is more complicated than in the single-view mode, and the inconsistency tolerance strategies should also be suitable for the multi-view mode Considering that the way of publishing changes and decision-making policies regarding tolerating inconsistency or intervening for resolution in multi-view cooperative systems are of great importance, therefore, they will be investigated as future works.

## References

[1] H. Klare, M. E. Kramer, M. Langhammer, D. Werle, E. Burger, and R. Reussner. "Enabling consistency in view-based system development—the VITRUVIUS approach." Journal of Systems and Software, 171, 110815, 2021.

[2] R. France and B. Rumpe. "Model-driven development of complex software: A research roadmap." In L. Briand and A. L. Wolf, editors, Future of Software Engineering, 37–54, 2007.

[3] J. Gausemeier, W. Schäfer, J. Greenyer, S. Kahl, S. Pook, and J. Rieke. "Management of cross-domain model consistency during the development of advanced mechatronic systems." In DS 58-6: Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 6, Design Methods and Tools. pages 1–12, 2009.

[4] H. Giese and S. Hildebrandt. "Incremental model synchronization for multiple updates." In Proceedings of the Third International Workshop on Graph and Model Transformations, GRaMoT'08, pages 1–8, 2008.

[5] J. Reineke and S. Tripakis. "Basic problems in multi-view modeling." In Tools and Algorithms for the Construction and Analysis of Systems, volume 8413 of LNCS, pages 217–232. Springer, 2014.

[6] M. Sharbaf, and B. Zamani. "Configurable three-way model merging." Software: Practice and Experience, 50(8), pages 1565-1599, 2020.

[7] H. Alfraihi, K. Lano, S. Kolahdouz-Rahimi, M. Sharbaf, and H. Haughton. "The impact of integrating agile software development and model-driven development: a comparative case study." In System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering. pages 229-245, 2018.

[8] J. Ralyté, R. Deneckère, and C. Rolland, "Towards a Generic Model for Situational Method Engineering", 15th International Conference on Advanced Information Systems Engineering, Klagenfurt/Velden, Austria, pages 95-110, 2003.

[9] M.R. Basirati, M. Zou, H. Bauer, N. Kattner, G. Reinhart, U. Lindemann, M. Böhm, H. Krcmar, and B. Vogel-Heuser. "Towards systematic inconsistency identification for product service systems." in Proceedings of the DESIGN 15th International Design Conference, 2018.

[10] Weidmann, Nils. "Fault-Tolerant Consistency Management in Model-Driven Engineering." PhD thesis, Paderborn University, 2021.