# Definition of Terms

FOR THE CLASSIFICATION FRAMEWORK OF THE PAPER

**Collaborative Model-Driven Software Engineering: Practices and Needs in Industry**.

2022 – v1.0

Table 1: Classification framework for Collaborative MDSE

| **MODEL MANAGEMENT (29)** |
|---|
| **Models and languages** |
| Collaboration at the model level |
| Collaboration at the metamodel level |
| Multi-view modeling (e.g. different views for different stakeholders) |
| Use of general-purpose modeling languages (e.g., UML, CAD) |
| Use of domain-specific languages |
| Import of an external language into the modeling environment |
| Projectional editing |
| **Model manipulation** |
| Model validation |
| Model execution |
| Model debugging |
| Model browsing/searching |
| Model testing by defining the test cases in the models |
| Lazy loading of the models/workspace |
| Round-trip engineering (from model to code and back) |
| Code generation |
| Model transformation |
| Integration with build/DevOps tools (e.g., cmake, Jenkins) |
| Database integration |
| Metrics of model complexity |
| Natural Language Processing (for model building) |
| **Editors and modeling environments** |
| Visual editors |
| Textual editors |
| Tabular editors |
| Tree-based editors |
| Sketch-based editors |
| Editors supporting multiple types of notations |
| Desktop-based modeling environments |
| Web-based modeling environments |
| Mobile device based modeling environments |
| **COLLABORATION (27)** |
| **Stakeholder management & access control** |
| Role-based access control |
| Authentication and authorization from corporate database |
| Anonymous access |
| User identification |
| User presence visualization |
| **Collaboration dynamics** |
| Human-Machine collaboration |
| Real-time collaboration |
| Offline (non-Real-time) collaboration |
| **Versioning** |
| Model differencing |
| Model differencing based on the modeling language, not on the file contents |
| Internal versioning support |
| External version control (for instance Git, SVN) |
| Model merging |
| Version branching |
| Undo-redo support during collaboration |
| History |
| **Conflicts and consistency** |
| Locking |
| Prevention of conflicts |
| Conflict awareness features (for instance, warnings, prompt actions) |
| Automation of conflict resolution |
| Manual conflict resolution |
| Metrics of degree of conflict/inconsistency |
| Eventual consistency |
| Push notifications on conflicts |
| **Network architecture & robustness** |
| P2P (serverless) network architecture |
| Cloud-based network architecture |
| Failure recovery |
| **COMMUNICATION (25)** |
| **Synchronous communication** |
| Chat |
| Audio |
| Voice |
| Hand gestures |
| Face-to-face |
| Change review sessions |
| Screen sharing |
| **Asynchronous communication** |
| Email |
| Wiki |
| Forum |
| Proposals |
| Voting |
| Annotations |
| Comments |
| Feedback |
| Reviews |
| Call-For-Attention |
| Sticky notes |
| Tags |
| Conflicts table |
| Multimedia annotations |
| Commit messages |
| Integrated professional-social networking |
| **Integration** |
| Communication means built into the modeling tools |
| Communication means NOT built into the modeling tools |

# 1 Model management

## 1.1 Models and languages

**Collaboration at the model level.** The collaborating parties are situated at the same level of abstraction.
**Collaboration at the meta-model level.** The collaborating parties are situated at different levels of abstraction. The subject of collaboration is a model one collaborator sees as a meta-* model.
**Multi-view modeling.** Decomposing models into different views where each view focuses on different aspect of the system.
**General-purpose modeling languages.** Modeling languages that are not specific to one domain, e.g., UML, CAD, etc.
**Domain-specific languages.** Modeling languages specific to one domain, typically with a restricted set of language elements. Examples of domain-specific languages include: AADL, SQL, GraphViz.
**Import of an external language into the modeling environment.** The collaboration mechanisms are not coupled with a specific modeling language, but rather external modeling languages can be imported into the workspace.
**Projectional editing.** Model edits are executed directly on the abstract syntax tree without a parser.

## 1.2 Model manipulation

**Model validation.** Evaluation of rules governing the well-formedness, well-typedness, and other validity concerns of the model.
**Model execution.** Augmenting the model with behavioral semantics and enacting it by means of software, typically simulation.
**Model debugging.** Discovering and fixing errors in the model using appropriate tooling, such as tracers, break points and dynamically-evaluated functions.
**Model browsing/searching.** Viewing and searching for models and their internal elements.
**Model testing by defining the test cases in the models.** Tests are defined by the primitives of the modeling language, such as graph queries in models following graph semantics, boundary conditions in models based on differential equations, etc. This is in contrast, for example, with testing EMF models in Eclipse using JUnit.
**Lazy loading of the models/workspace.** Loading models only when needed, typically to improve performance and save resources.
**Round-trip engineering.** Mechanism to keep artifacts in sync that are involved in forward- and backward engineering processes. For example, the synchronization of models and the source code generated from them upon a change either in the models or in the source code.
**Code generation.** Forward engineering approach that produces source code from models.
**Model transformation.** Operation that takes as input one or more models and either modifies them or creates new models out of them. Model transformations can be categorized based on e.g., the metamodels the models conform to (endogenous vs exogenous transformations), the output artifact (in-place vs out-place transformations), their direction (uni-directional vs bi-directional transformations), or their execution semantics (batch vs incremental/reactive transformations).
**Integration with build/DevOps tools.** Integration of the modeling facilities with tools of continuous integration, testing and deployment to regularly and automatically inspect models, catch regressions, and deploy them to test/production.
**Database integration.** The ability of the modeling tool to persist models in, and load models from a database service.
**Metrics of model complexity.** Numeric and categorical information about the complexity of the models that are suggestive of maintainability and inconsistency-proneness.
**Natural Language Processing (for model building).** Automation of the development of formal models (such as graph models) from informal models (such as textual requirements) by means of Natural Language Processing.

## 1.3 Editors and modeling environments

**Visual editor.** Editors providing visual/graphical means of model development, leveraging information such as shapes, colors, size, topology, etc.
**Textual editor.** Editors providing textual means of model development.

**Tabular editor.** Editors using tables as the primary means of model development.
**Tree-based editor.** Editors using tree hierarchies as primary means of model development.
**Sketch-based editor.** Editors that allow formal modeling by free-hand sketching. Typically, pattern recognition is used in the background to interpret sketches and map them onto formal structures.
**Editors supporting multiple types of notations.** Single editor providing support for more than one notation, e.g., graphical and textual notation for the same model.
**Desktop-based modeling environments.** Modeling environments that can be installed/used on a desktop or laptop computer.
**Web-based modeling environments.** Modeling environments that can be accessed/used through a browser.
**Mobile device based modeling environments.** Modeling environments ready to be used on hand-held devices.

# 2 Collaboration

## 2.1 Stakeholder management & access control

**Role-based access control (RBAC).** The rights to access models and specific model elements are grouped into roles, and stakeholders are associated with those roles.
**Authentication and authorization from corporate database.** User credentials are stored in and verified from company-specific databases, such as an active directory.
**Anonymous access.** Allowing unidentified users to access models for reading or modifications.
**User identification.** Using logical entity to identify a user for accessing models.
**User presence visualization.** Providing visualization for the presence of other users collaborating on the same model(s).

## 2.2 Collaboration dynamics

**Human-Machine collaboration.** Collaboration between human and machine, such as cobots and chatbots.
**Real-time collaboration.** Collaboration model in which users can view and edit models at the same time and observe each others' changes immediately.
**Offline (non-Real-time) collaboration.** Collaboration model in which changes are persisted to be observed by other collaborators substantially later. Typical implementations include versioning systems, such as Git, SVN, etc.

## 2.3 Versioning

**Model differencing.** Operation that calculates the difference between two models, typically between two different snapshots of the same model for reasoning about its changes.
**Model differencing based on the modeling language, not on the file contents.** Operation that calculates the difference between two models by considering the model's concepts as the primitives, instead of the raw data behind it.
**Internal versioning.** Versioning mechanism that expresses model versions by model concepts.
**External version control.** Versioning mechanism that expresses model versions by serialized data, such as textual information. Often, textually serialized models are versioned in source code versioning systems, such as Git.
**Model merging.** Operation that calculates the join of two models, often specified as a set union or lattice join operation.
**Version branching.** Mechanism for storing multiple alternative versions of the same base model.
**Undo-redo support during collaboration.** Mechanism to revoke previously executed operations in a FILO fashion. Can be local or distributed.
**History.** Mechanism to serialize the list of model changes by an ordering operator, typically the timestamp of the operation. Real-time distributed modeling systems might opt for different ordering operators, e.g., insert sequence codes.

### 2.4 Conflicts and consistency

**Locking.** Mechanism that prohibits edit operations on models by collaborators other than the one who acquired the lock.
**Prevention of conflicts.** Mechanism that prevents collaborators to perform conflicting changes on the models before they happen.
**Conflict awareness.** Functionality that allows collaborators to be aware of conflicting information or changes in the collaborative setting.
**Automation of conflict resolution.** Mechanisms that aid the collaborators in resolving conflicts. Often implemented as a combination of diff and merge mechanisms.
**Manual conflict resolution.** Resolution of conflicts by the collaborators manually without the aid of the computer.
**Metrics of degree of conflict/inconsistency.** Numeric and categorical information about how much the models are conflicting or inconsistent with each other. The metrics are suggestive the effort needed to resolve the conflicts.
**Eventual consistency.** Mechanism that guarantees the liveliness property in a distributed setting, that is, local replicas will eventually observe the same set of update messages.
**Push notifications on conflicts.** Mechanism to inform collaborators about newly-created conflicts.

### 2.5 Network architecture & robustness

**P2P (serverless) network architecture.** Peer-to-peer. Architecture without a central mediating or orchestrating entity.
**Cloud-based network architecture.** Architecture in which models and/or tools are stored in a public or private cloud platform.
**Failure recovery.** Mechanism to retain the latest functional versions of the models upon unexpected shut-downs.

## 3 Communication

### 3.1 Synchronous vs Asynchronous communication

**Synchronous communication.** Communication model in which a communicating party has a reasonable belief that the other communicating party receives a message and processes it near real-time. Examples include chat, call, video call, screen sharing, etc.
**Asynchronous communication.** Communication model in which a communicating party cannot assume that the the other communicating party will receive or process a message within a time frame that would be considered near real-time. Examples include emails, text messages, wiki pages, commit messages, etc.

### 3.2 Integration

**Built-in communication features.** Communication features that are part of the modeling tool.
**External communication features.** Communication features that are not part of the modeling tool, but are used in the collaborative endeavor regardless. Typical examples in 2021 include Microsoft Teams, Slack, Google Chat, etc.