

- Modeling Workbench
 - **Notation:** The notation of the modeling language.
 - **Textual:** The modeler edits the model through text.
 - **Symbols:** The modeler can use symbols such as mathematical notations.
 - **Graphical:** The modeler edits the metamodel through different graphical representations such as diagrams.
 - **Tabular:** The modeler edits the metamodel through a table.
 - **Tree:** The language designer edits the metamodel through a tree editor.
 - **Block:** The modeler edits the metamodel through a block representation like [Blockly](#).
 - **Form:** The modeler edits the metamodel through a form representation.
 - **Semantics:** Features concerning the model semantics
 - **Translational:** Compilation to a program expressed in another language.
 - **M2T:** Compilation from a model of the developed language to the text of another language.
 - **M2M:** Compilation from a model of the developed language to the model of another language.
 - **Interpretative:** Direct execution by the host language without prior translation.
 - **Editor:** Available modeling workbench's editor features.
 - **Editing mode:** How the metamodels are edited.
 - **Free-form:** The modeler freely edits the persisted model.
 - **Representation:** The modeler edits a representation of the model and both are persisted. The representation does not necessarily have a fixed layout.
 - **Projectional:** The modeler edits a projection of the persisted model in a fixed layout.
 - **Syntactic services:** Model workbenches syntactic services.
 - **Highlighting:** Visually distinguishes syntax elements of metamodels in the editor using colors and styles to improve readability.
 - **Outline:** Displays a structured, hierarchical view of a metamodel's components to aid navigation.
 - **Folding:** Allows collapsing and expanding sections of metamodels based on structural elements to improve focus and readability.
 - **Syntactic completion:** Suggests possible completions for metamodel elements based on syntax rules.
 - **Diff:** Compares different versions of a metamodel, highlighting added, removed, or modified parts.
 - **Auto-formatting:** Automatically adjusts indentation, spacing, layout, and structure according to predefined style rules.
 - **Semantic services:** model workbenches semantic services.

- **Reference resolution:** Identifies and links metamodel elements to their declarations or definitions.
- **Semantic completion:** Provides context-aware suggestions by analyzing the meaning of metamodel parts.
- **Refactoring:** Supports automated metamodel transformations (e.g., renaming, extracting parts) to improve maintainability without altering functionality.
- **Error marking:** Detects and highlights syntactic or semantic issues in the metamodel, often with tooltips explaining the problem.
- **Quick fixes:** Suggests and applies automated solutions for detected issues.
- **Origin tracking:** Keep track of metamodel's elements during the different transformation steps. Useful for error displays.
- **Live translation:** Capability to use the designed model during its development.
- **Views:**
 - **Debugger:** A dedicated view for debugging, e.g., buttons for setting breakpoints, going into, forward.
 - **Call Hierarchy:** A dedicated view for seeing the hierarchy of past calls, similar to method call hierarchy in IDEs for GPLs
 - **Model Hierarchy:** A view for observing inheritance trees of hierarchical models. Similar to class hierarchy view in OO GPLs
- **Viewpoint management:** How the different models are presented to the modeler.
 - **Multi-views:** Capability to propose different viewpoints over the whole modeling workbench.
 - **Blended modeling:** Capability to propose different notations for a single model.
- **Validation:** Features concerning the validation of a model made with the developed language.
 - **Syntactic check:** Validation of the structure of the model (syntaxes).
 - **Naming:** Name binding
 - **Types:** Type systems
 - **Formal verification:** Capability to prove parts of the model definitions, through a compilation to Coq for instance.
 - **Data Flow Analysis:** Analysis for data flow within models, e.g., to detect cyclic dependencies or deadlocks.
 - **Test model generation:** Capability to generate models conform to a language definition, allowing the modeler to have first examples.
- **Testing:** Features to help the modeler verify their models.
 - **Model debugging:** The modeling workbench provides a debugger to debug some model definition concerns.
 - **Omniscient debugging:** Capability of a debugger to go backward in addition to forward.
 - **Model testing:** The modeling workbench provides ways to unit-test models.

- **Collaboration:** Features specific to the collaboration between different modelers.
 - **Live collaboration:** Collaboration at the same moment in time.
 - **Strategy:** How the live collaboration is done inside the modeling workbench.
 - **Optimistic:** Model designers can edit the same model or even the same element at the same time. Requires modification merge strategy.
 - **Pessimistic:** Editing an element causes it and possibly its related items to be locked.
 - **Collaboration architecture:** In technical terms, how the collaboration is done among the different modelers clients.
 - **Distributed:** Each client is independent and can work offline. The data are exchanged among the different clients (e.g., Git or CRDT).
 - **Centralized:** A central server is required to manage and control the collaboration (e.g., SVN).
 - **Versioning:** The modeling workbench proposes an integrated way to version developed languages.
- **Architecture:** Features concerning the architecture of the modeling workbench.
 - **Platform:** On which kind of platform does the modeling workbench run?
 - **Desktop:** The backend and frontend of the language workbench cannot be uncoupled and are both directly executed on the language designer device.
 - **Cloud-native:** The backend and frontend of the language workbench are uncoupled and may be executed on different devices.
 - **Modular:** The modeling workbench is thought to be extended, proposing APIs.