Vocabulary

Modeling workbench: environment proposing different tools to facilitate the use of one or several languages. “Synonym”: IDE.
End user: person using a modeling workbench to develop models.
Model: Artifact developed by an end-user using languages. “Synonym”: program.
[Language] service: “language specific functionality that can be used by a language user *[a/n: end user]* and that takes a program *[a/n: model]* as input. Such service can be a functionality related to editing programs giving quick feedback to the user (e.g., auto completion, goto definition, etc) but can also be a more long running functionality related to other activities (e.g., compiler, debugger, etc).” [1]

[1] Fabien Coulon, Alex Auvolat, Benoit Combemale, Yérom-David Bromberg, François Taïani, Olivier Barais, and Noël Plouzeau. 2020. Modular and distributed IDE. In Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2020). Association for Computing Machinery, New York, NY, USA, 270–282. https://doi.org/10.1145/3426425.3426947

Feature definition

- Modeling Workbench
    - **Notation:** The notation of the modeling language.
        - **Textual:** The end-user edits the model through text.
            - **Symbols:** The end-user can use symbols such as mathematical notations.
        - **Graphical:** The end-user edits the model through different graphical representations such as diagrams.
        - **Tabular:** The end user edits the model through a table.
        - **Tree:** The end user edits the model through a tree editor.
        - **Block:** The end user edits the model through a block representation like Blockly.
        - **Form:** The end user edits the model through a form representation.
        - Notation Paradigm
            - **Internal:** The language is presented as an internal language of another language such as a GPL. “using a host language to give the host language the feel of a particular language.” [1]
                - **Fluent API:** Method of designing an API to allow method chaining, making code more readable and expressive.
                - **Shadow embedding:** Capability to embed in the host language custom syntaxes. For instance, TSX from React embedding HTML concepts.
                    - **Specialization:** Capability to specialize some concepts of the host language for the internal DSL.

            - **External:** The language is presented as an external language, with its own syntax uncoupled from the host language.
    - **Semantics:** Features concerning the model semantics
        - **Translational:** Compilation to a program expressed in another language.
            - **M2T:** Compilation from a model of the language to the text of another language.
            - **M2M:** Compilation from a model of the 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 models are edited.
            - **Free-form:** The end-user freely edits the persisted model.
            - **Representation:** The end-user edits a representation of the model, and both are persisted. The representation does not necessarily have a fixed layout.
            - **Projectional:** The end-user edits a projection of the persisted model in a fixed layout.
        - **Syntactic services:** Model workbenches syntactic services.
            - **Highlighting:** Visually distinguishes syntax elements of models in the editor using colors and styles to improve readability.
            - **Outline:** Displays a structured, hierarchical view of a model's components to aid navigation.
            - **Folding:** Allows collapsing and expanding sections of models based on structural elements to improve focus and readability.
            - **Syntactic completion:** Suggests possible completions for model elements based on syntax rules.
            - **Diff:** Compares different versions of a model, 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 model elements to their declarations or definitions.
            - **Semantic completion:** Provides context-aware suggestions by analyzing the meaning of model parts.
            - **Refactoring:** Supports automated model transformations (e.g., renaming, extracting parts) to improve maintainability without altering functionality.
            - **Error marking:** Detects and highlights syntactic or semantic issues in the model, often with tooltips explaining the problem.
            - **Quick fixes:** Suggests and applies automated solutions for detected issues.
            - **Origin tracking:** Keep track of model'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 end-user.
            - **Multi-views:** Capability to propose different viewpoints over the whole defined system.
            - **Blended modeling:** Capability to propose different notations for a single model.
    - **Validation:** Features concerning the validation of a model.
        - **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, allowing the end-user to have first examples.
    - **Testing:** Features to help the end-user 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 end-users.
        - **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 end-users 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 models.
- **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 modeling workbench cannot be uncoupled and are both directly executed on the end-user device.
        - **Cloud-native:** The backend and frontend of the modeling workbench are uncoupled and may be executed on different devices.
    - **Modular:** The modeling workbench is thought to be extended, proposing APIs.