MetaModules

Web-based Tools for (Domain Specific) Language Engineering

LangDev 2022

Jürgen Mutschall

Aachen, 26.09.2022

Agenda (30min)

Web-based Tools for (Domain Specific) Language Engineering

1. Deriving a tool architecture from the „Open Platform“ requirements
2. Design decisions and system architecture
3. Featured Aspect: Blended Modeling and Projectional Editing
4. Live demo of some tool functionality under development

An Open Platform for Systems and Business Engineering Tools

“A platform for developing systems and business engineering tools must be unbiased regarding the languages used to define the models and the analyses that run on them. Instead it should provide the infrastructure for defining languages, for working effectively with large, multi-paradigm models in a collaborative manner, and for integrating arbitrary model analysis and transformation services.”

(Markus Voelter, December 2019)

Angabe Bildquelle

Requirements List

Summary of the requirements in the whitepaper by Markus Voelter

End User Perspective

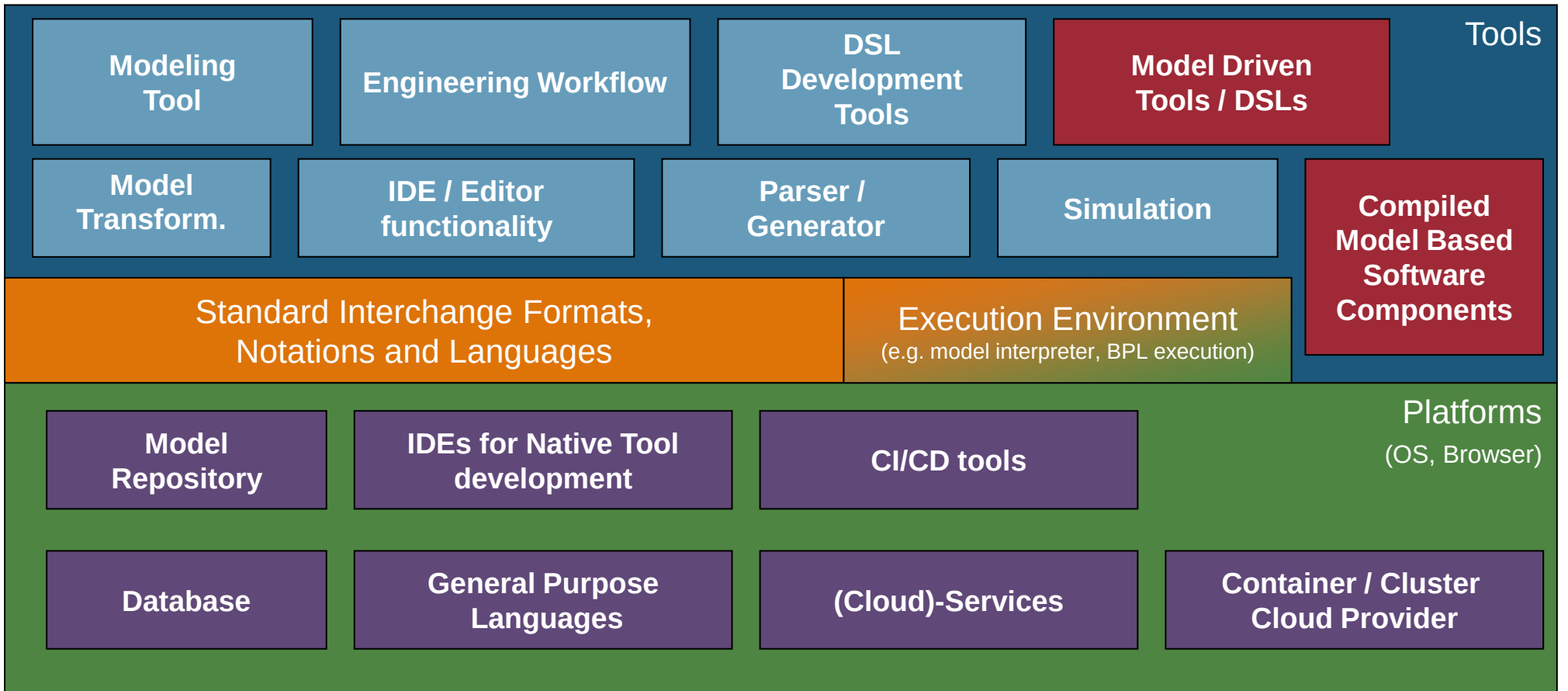
– Scalable
– Collaboration and Versioning
– Migration Support
– Roles, View and Contexts
– Native to the Web
– IDE Features
– Liveness
– Growable
– Continuous Integration

Language Engineering Perspective

– Multi-Paradigm
– Multi-Notational
– Language Composition
– External Tools

Logical Architecture

Tools and tools to develop tools …



Requirements List+

Extended Summary of the Requirements

End User Perspective

– Scalable
– Collaboration and Versioning
– Migration Support
– Roles, View and Contexts
– Native to the Web
– IDE Features
– Liveness
– Growable
– Continuous Integration

Language Engineering Perspective

– Multi-Paradigm
– Multi-Notational
– Language Composition
– External Tools

Platform Engineering Perspective

– Container- and Cluster-Support
– “State of the Art”-Development Tools
– OS-independent, but Compilation to Native Code
– Meet Expectations in Terms of User Experience

Findings from Previous Developmental Approaches

“Empiricism is wiser than wishful thinking” or “You learn from your mistakes!”

– The most successful tools have a very low barrier to entry.
– The user acceptance of an extension for an existing, well known tool is higher than a new tool and/or workflow.
– A developer is also a user. The user prefers “good-looking” tools.
– **The user experience is becoming more and more important.**

– DSL engineering, especially the definition and validation of type systems, is hard.
– Only few developers are experts in DSL engineering, many developers design a DSL almost “once in a lifetime”.
– A lot of DSLs are simple configuration languages without advanced typing/semantics (JSON with “syntactic sugar”)
– **Reuse and composition of DSLs is becoming more and more important.**

– There have been some major failures in the establishment of standards for graphical modeling languages/notations, constraint and transformation languages, execution semantics and model interchange formats.
– **But without any agreement on common standards a tool platform cannot be “open”.**

– Graphical modeling tools/notations are easy to use, but it is difficult and very time-consuming to design good/complete models, which can be used for code generation or simulation.
– It is very difficult to find the correct balance between expressiveness and practical usability.
– Complex constraints, relations, generic types and annotations are more easy described in a textual language.
– **Blended modeling is the future.**

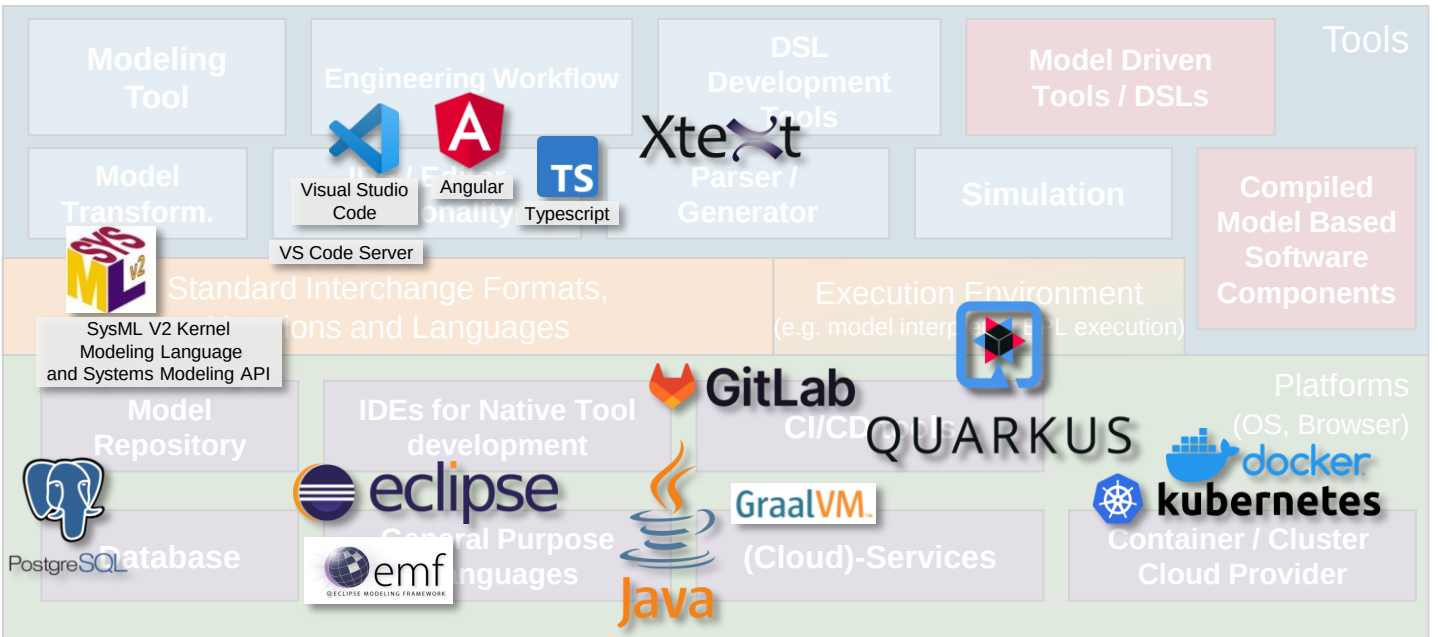
Design Decisions

Balance of Best Practices and New Approaches

Client
– Completely browser based client software
no client installation
– Typescript, Microsoft VS Code and Angular
as client technology

Server
– Platform services based
on Quarkus Microservice Architecture
– Modeling services (e.g. transformation,
language server) implemented in Java;
compiled to the metal by GraalVM

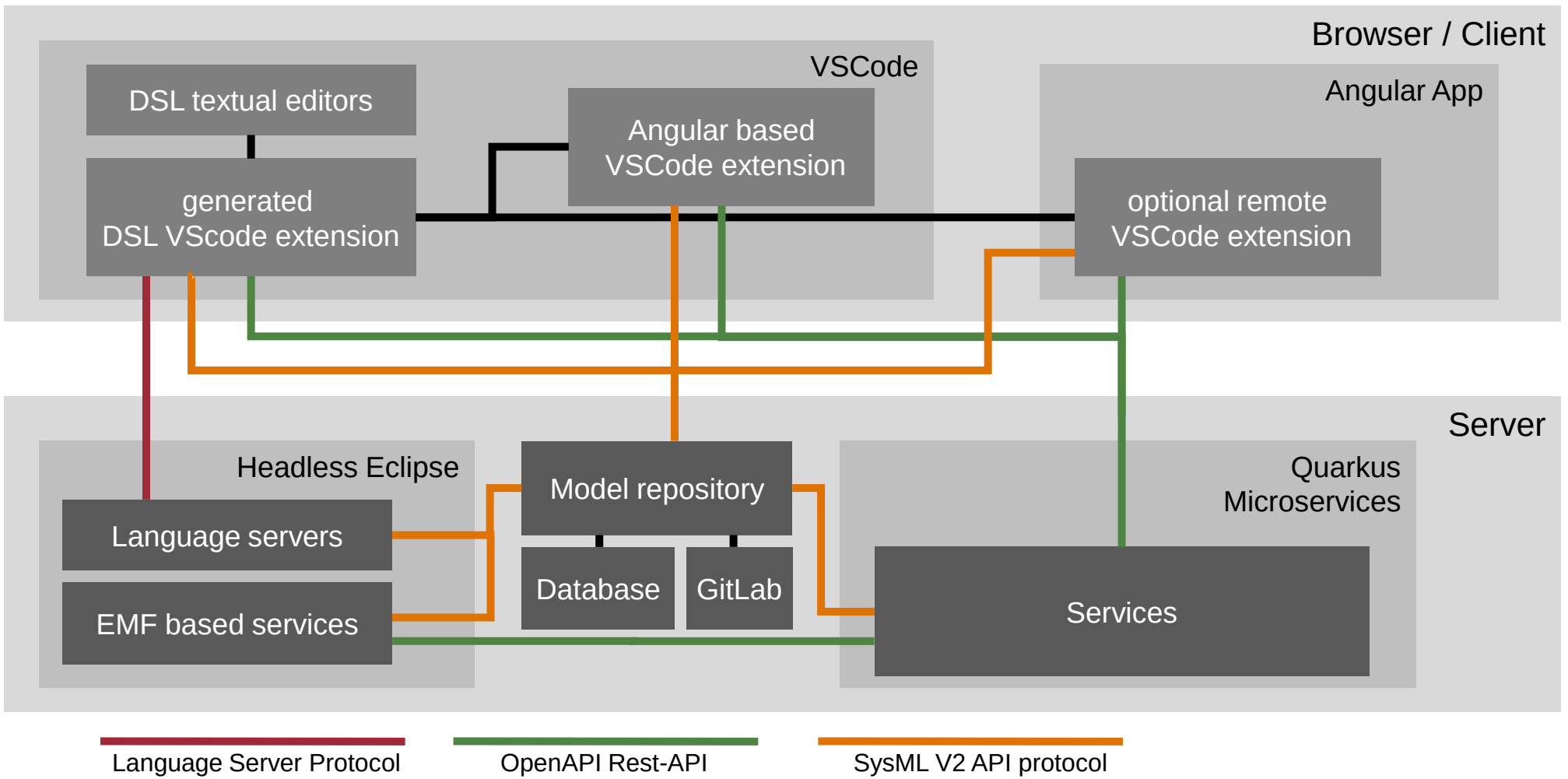
– Services running in Kubernetes / Docker;
cluster support and cloud provider ready
  – MS VSCode server installation / workspaces
  – Headless Eclipse / workspaces / Xtext / EMF
  – Gitlab CI/CD pipeline / versioning



– Model repository based on SysML V2 standard.
Using standard SQL database for scalability and performance reasons.
– SysML V2 Kernel Modeling Language as interchange format

Interfaces / Networking
– SysML Systems Modeling API for remote access to the model repository
– OpenAPI based REST/websocket interfaces for Quarkus remote access
– (~~Graphical~~) Language Server Protocol

Architecture



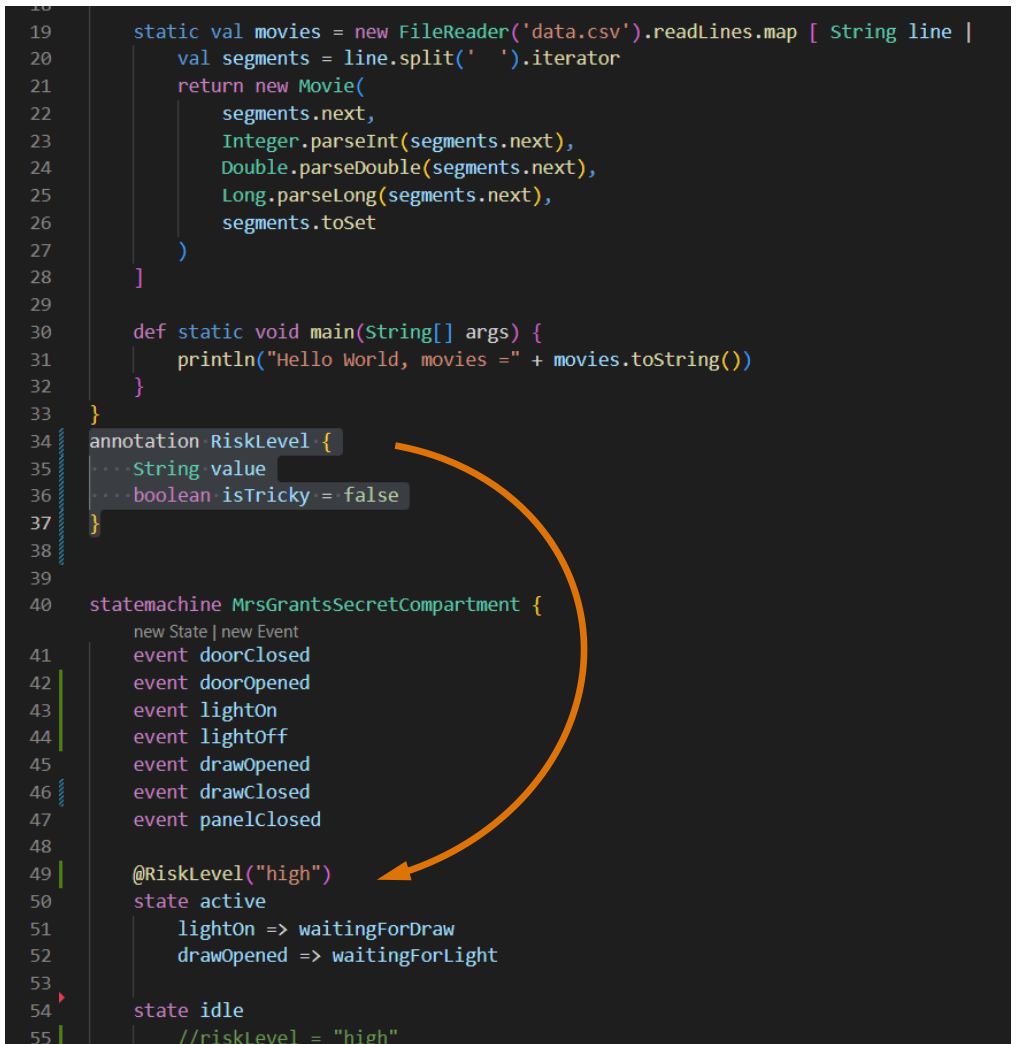
Xtext-based VSCode editor for multi-paradigm Source Editing

DEMO

- Multiple DSLs can be mixed as needed.
- Working on one common model
- In this example a simple statemachine DSL is mixed with a general programming language (GPL).
- The annotation, defined in the GPL is used to annotate the simple “state” entity of the statemachine DSL.
- The code generator for the statemachine DSL delegates the annotation aspect to the code generator for the GPL.

GPL

Statemachine

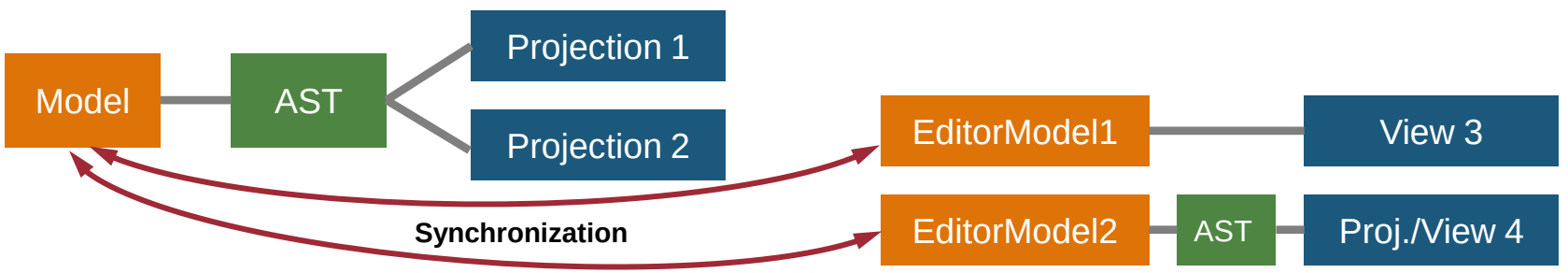


Excurse: Projectional Editing and Blended Modeling

Integration of graphical and textual (and multi-paradigm) modeling

*"**Projectional Editing** (as an alternative to Source Editing) **i**s the idea that the core definition of a system should be held in a model and edited through projections. … With projectional editing the abstract representation is the is core definition of the system. A tool manipulates the abstract representation and projects multiple editable representations for the programmer to change the definition of the system."*

(Martin Fowler, 2008)



*"**Blended modeling** means to allow engineers to freely choose and switch between several different notations for the same domain-specific concepts captured in a DSML. Traditionally, DSML tools focus on one specific notation (such as text, diagrams, tables or forms). This limits human communication, especially across engineering disciplines. A notation that is well-understood by one engineering discipline may not be understood by engineers from another discipline. Moreover, engineers (from the same or different disciplines) may have different notation preferences; not supporting multiple notations negatively affects engineers' throughput."*

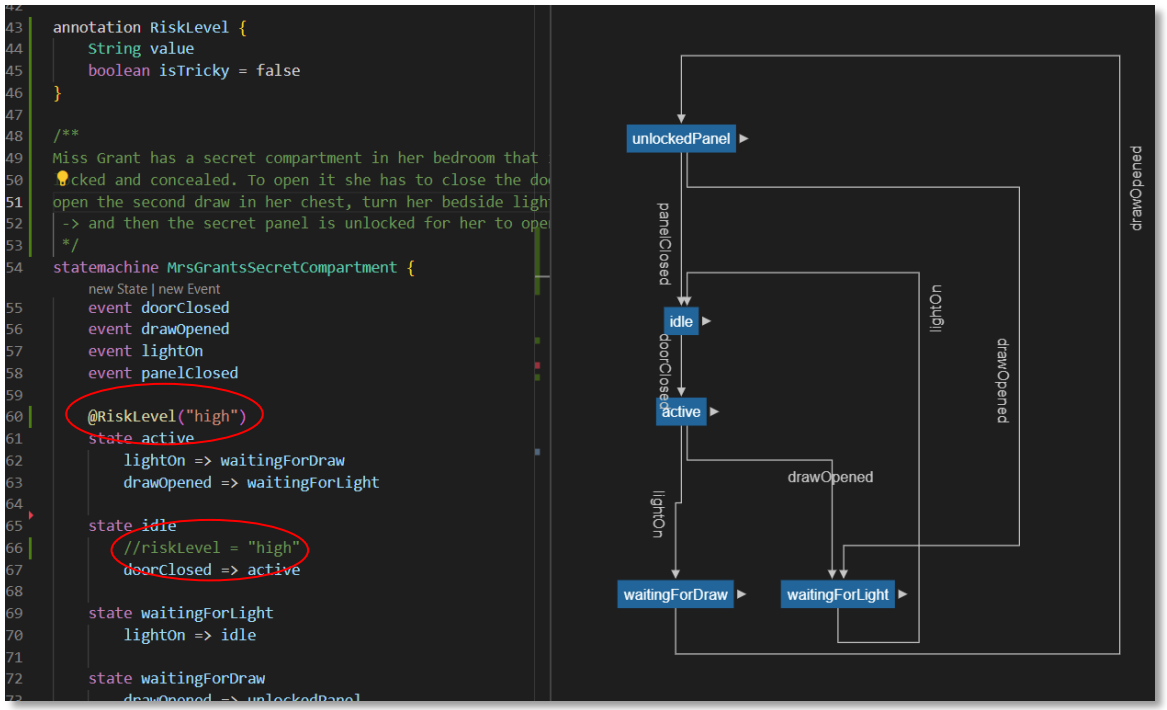
(ITEA3 EU project BUMBLE, 2019**)**

Blended Modeling is More General than the Projection of an AST

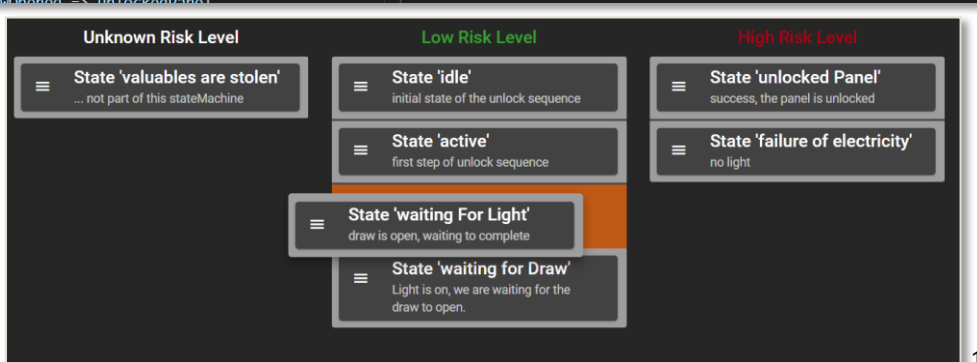
DEMO: The goal is to mix arbitrary UI elements, graphical structure and textual code

Simple example:

A model entity has an attribute “risk level” which has one of the three states *“unknown”, “low”* or *“high”.* In a textual editor the user would edit the risk level attribute by changing the specific line in the code editor (annotation or property). The syntax autocompletion of the editor may reduce the count of keyclicks to a minimum.



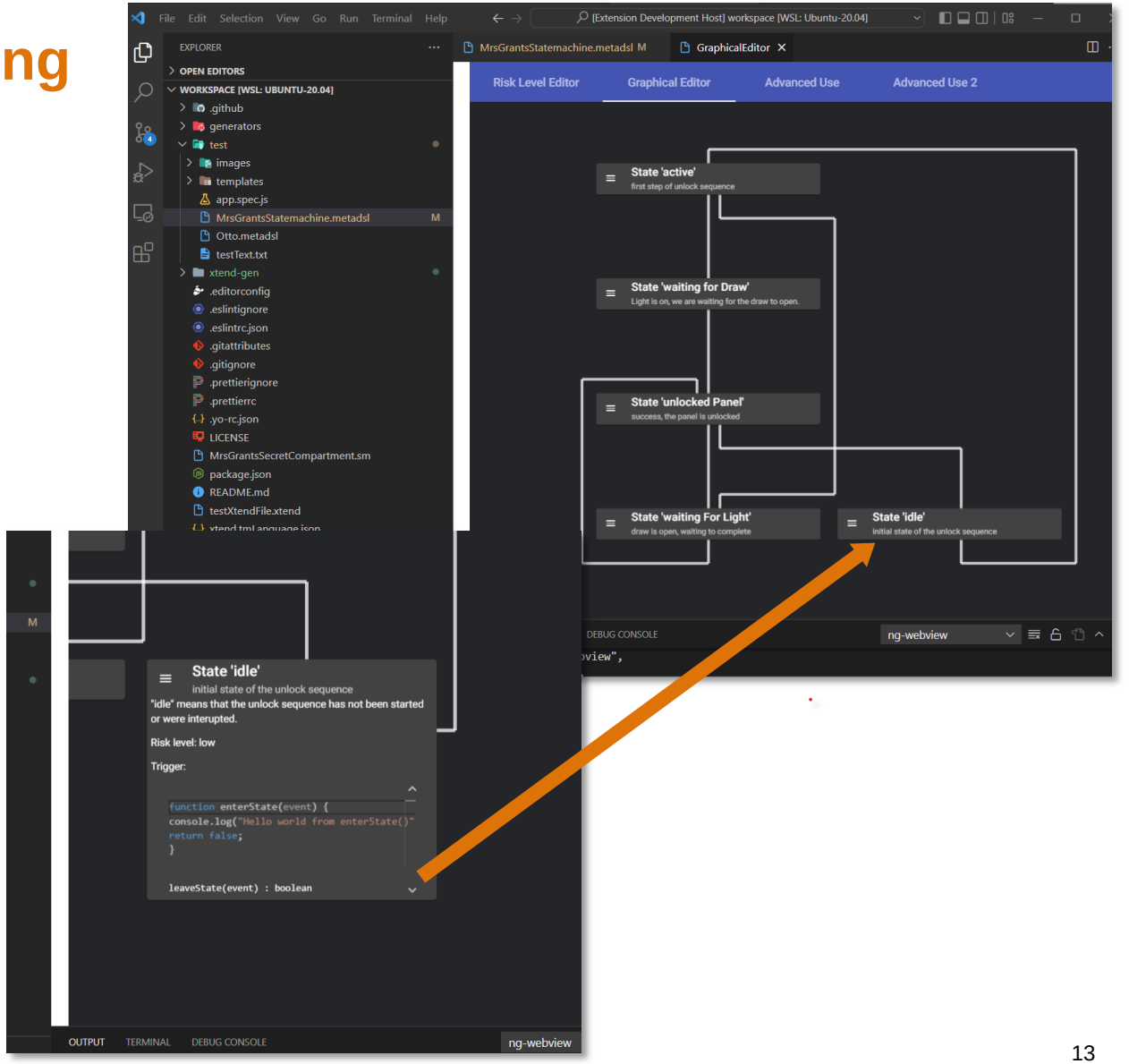
This is ok, if the attribute is seldom changed and the count of entity instances is small.
But if the user wants to do an risk assessment / adjustment on a global level, e.g. comparing entities, (s)he would prefer a specific user interface or editor to do this. This drag’n-drop editor is not a “projection” of an AST.
The columns of the table editor are the possible values/enum of the annotation of entities.



Elements for Blended Modeling

DEMO

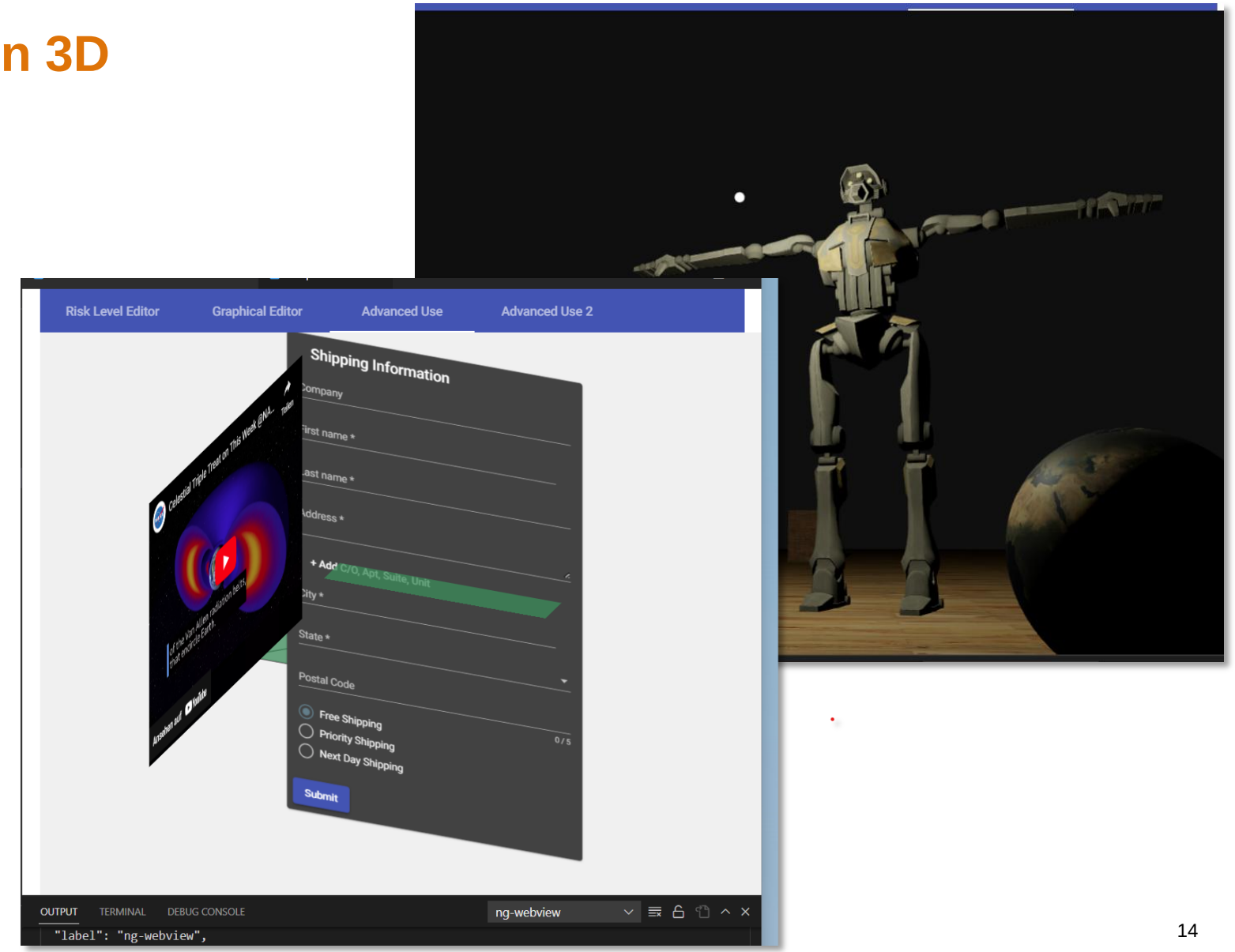
- Goal: Replacing the SVG-Sprotty-based editor by an Angular based graphical editor with embedded Monaco code editor
- All features of the VSCode editor environment (autocompletion, code lenses, etc.) embedded in an expandable angular component integrated in a graphics component.
- And even more ...., see next slide



Blended Modeling in 3D

DEMO

- The graphical editor is part of a powerful 3D environment.
- Every user interface can be integrated into a 3D visualization, e.g. a manufacturing line, digital twin of a city, etc ....



MetaModules

Vielen Dank!

@J_Mutschall
juergen.mutschall@desys.com