Modeling-Based Project Planning/Documentation

This presentation was created by *Rubén Senae Winzer Ortega* as part of a seminar for our study project 'StayInSync'.

Overview: Modeling and Objective of this Seminar

Modeling plays a key role in understanding, designing, and communicating complex systems and can be seperated into two main categories:

- **Descriptive modeling** captures the current state the system is in.
- **Prescriptive modeling** defines the future state of the system.

There are multiple established **modeling languages**, ranging from domain-specific to general-purpose, as well as a wide range of **tools** that support their creation, use, and integration into the development process.

The goal of this seminar is to find the modeling language as well as tools supporting that suits our Use Case The Best:

The development of a **System of Systems** (SoS)/**System as a Service** (SaaS) called "***StayInSync***" *for* "**Robert Bosch GmbH**", to be developed over a period of six months by a team of 11 students of the **University of Stuttgart**.

Benefits of Modeling for Developers and Customers

Modeling provides various **advantages** for both developers and customers, improving the overall development process and product quality .

Pros for Developers:

- **Clarity**: Models help developers better understand the system structure.
- **Connectivity**: Planning and working with intern interfaces requires a clear understanding of relations and is supported by models
- **Error detection**: Potential issues can be identified early.
- **Better collaboration**: Enhances communication within the team.
- **Documentation**: Records important design decisions.

Pros for the Customer:

- **Clearer communication**: Helps customers better understand the system.
- **Early feedback**: Customers can provide input early in the process.
- **Expectation management**: Improves alignment between expectations and deliverables.
- **Risk reduction**: Minimizes the risk of misunderstandings or misdevelopment.

Even if developers don't see modeling as essential for their own work, creating descriptive models to address customer needs can still be highly beneficial for the final acceptance of the product

StayInSync: Classification of our Use Case

To find **suitable modeling languages** and supporting **tools** for our **Use Case**, we first need to **analyze** the nature of the system and its requirements as well as the needs of our Developers and Customer.

Developer perspective:
We are a small team of students with limited time and resources. As a result, **prescriptive modeling** is likely to be **used sparingly**. However, since we are working in parallel on many interconnected microsystems (SoS), **descriptive models** are **essential** to effectively discuss and align interfaces across teams.

Customer perspective:
Even though our System is a **SaaS**, it won't be continuously maintained by our team after delivery. Therefore, it **must be easily understandable** and **maintainable by the customer**. Descriptive models can help ensure transparency and ensure a smooth handover and long-term usability.

Conclusion:
For our use case, the most **suitable modeling language** should be **easy to learn** for all stakeholders and **offer clear, expressive support for descriptive modeling**. Therefore, tools should focus on simplifying model generation from code and making traceability of the models as easy as possible.

What Makes a Good Model in a SoS Context?

Descriptive and **prescriptive** models need to bring certain criteria of the modeled **SoS** across:

- **Capture Autonomy**
  Models should show that subsystems operate independently
- **Reflect Dynamic Structure of Connectivity**
  Models must express that interfaces and communication paths may emerge, change, or vanish over time.
- **Enable Insight into Emergent Behavior**
  Models should help detect, explain, and evaluate unforeseen interactions
- **Allow for Evolution and Refinement**
  Models should support versioning, modular growth, and iterative updates as the SoS develops.
- **Ensure Alignment with Shared Objectives**
  Models should clarify how each subsystem contributes to the overarching goals of the SoS, even when local goals differ.

Evaluation of different Modeling Languages

DSML

Object Management Group®
https://www.omg.org/bpmn/

Object Management Group®
https://www.omg.org/uml-directory/

Object Management Group®
https://www.omg.org/sysml-directory/

Object Management Group®
https://github.com/systems-modeling

Domain-Specific-Modelling-Language (DSML)

DSML is a **highly specific** Modelling Language, individually created for certain domains and applications.

Pros:

- Offers high expressiveness for target domain
- Focus on relevant concepts reduces time waste with unnecesary constructs
- Can increase efficiency and clarity in communication among domain experts

Cons:

- High initial effort to design and implement the DSML
- Steep learning curve for all stakeholders
- Limited Tooling Support
- If poorly designed, a DSML can introduce project risks that are unique to domain-specific approaches

Success Factors	Positive Impact	Negative Impact
Domain expertise	**Good knowledge of domain concepts, vocabulary and terminology => Expressive DSML**	**Incomplete domain knowledge => Inexpressive DSML, lacking functionalities**
Domain scoping	**Fitting business needs**	**Domain too broad or too narrow**
Effective supporting tools	**Faster, better and cheaper DSML development**	**Costly DSML**
Effective meta-model	**Easy definition and upgrade of expressive, high-level abstract DSML.**	**Ambiguity Poor semantics**
Effective underlying generator	**Better exploitation of DSML models**	**Models exclusively intended for documentation purposes**
Domain engineering environment	**Specialized and dedicated teams for DSML definition**	**Weak domain expertise**
High level of abstraction	**Close to the real-world separation of concerns**	**Platform-dependent DSML**
Language expertise	**Coherent models, useful and non-redundant functionality.**	**Incoherent models' useless, redundant functionalities**
View point orientation	**Specialized DSMLs**	**Some stakeholders may be neglected**
Purpose-orientation	**Focused DSMLs**	**Incoherent models**
Domain expert support	**More support and fast adoption**	**Resistance and sabotage**
Effective DSML definition process	**Well-established practices for DSML definition**	**Individual approaches' unintended results**

Business Process Model and Notation (BPMN)

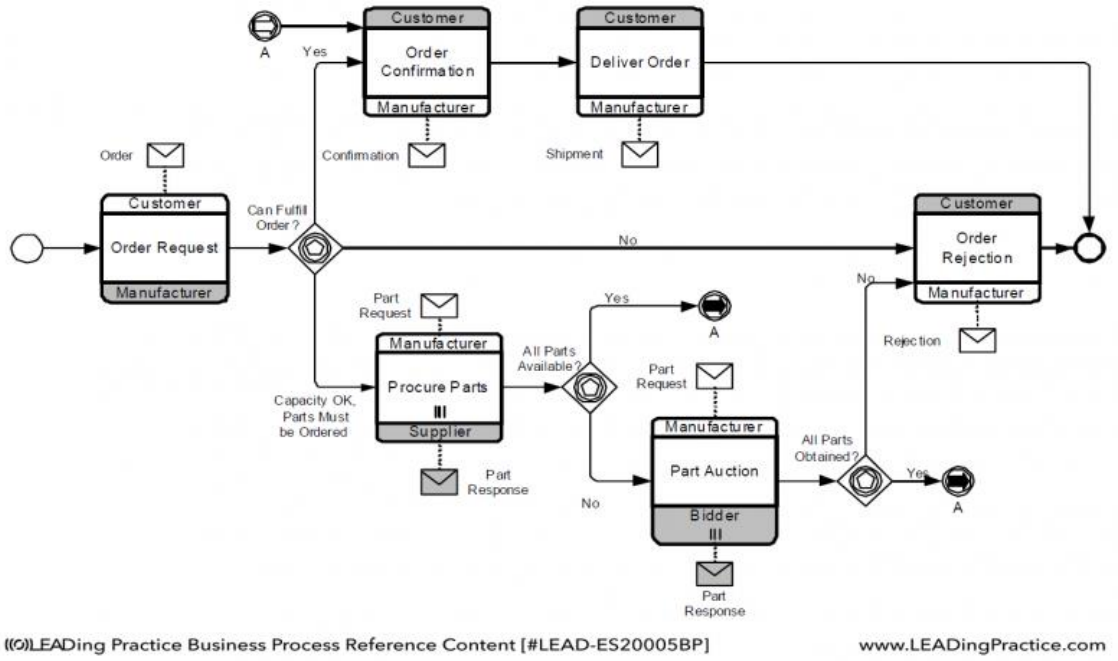
BPMN is a **standardized** modeling language for visualizing business workflows and organizational processes.

Pros:

- Intuitive for both technical and non-technical stakeholders
- Usefull for describing workflows and interactions

Cons:

- Not suitable for modeling technical structures like data models or system architecture
- Due to their simplicity in technical aspects, our customers may not find the models helpful when trying to maintain or further develop the service

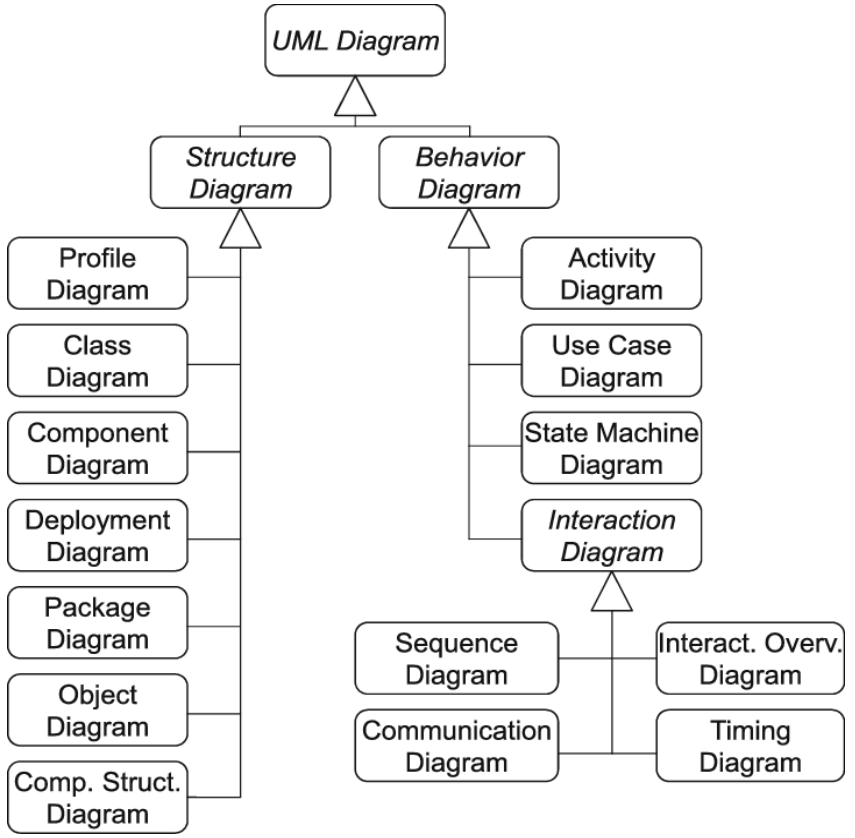


Unified Modeling Language (UML)

UML is a highly general-purpose modeling language that supports a wide range of modeling needs by offering multiple complementary "views" of a system.

Pros:
- Widley adopted -> strong community/tool-support
- Offers multiple standardized views, covering a big range of modeling needs
- Easy to learn at a basic level

Cons:
- Its generality can introduce unnecessary complexity to our domain
- Risk of inconsistencies between different views if not properly synchronized



Kautz, Oliver & Roth, Alexander & Rumpe, Bernhard. (2018). Achievements, Failures, and the Future of Model-Based Software Engineering. 10.1007/978-3-319-73897-0_13. , License: CC BY 4.0, https://www.researchgate.net/publication/325747903_Achievements_Failures_and_the_Future_of_Model-Based_Software_Engineering

Systems Modeling Language Version 1 (SysML v1)

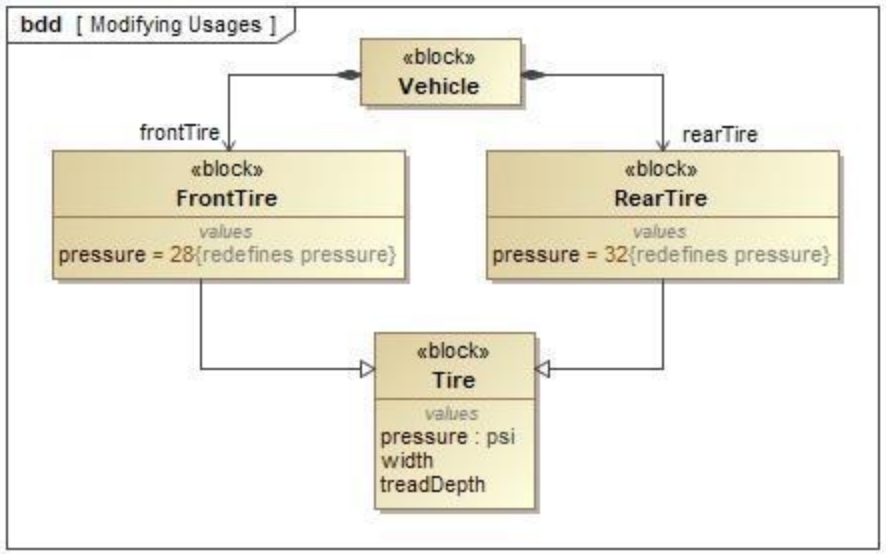
SysML v1 is based on UML but improved for systems engineering by reducing unnecessary UML elements and adding specialized diagrams

Pros:

- Extends UML with system engineering focus, adding Requirement and Parametric diagrams
- Compared to UML much more sized down on the most important diagrams
- Transformation to SysMLv2 would be possible if necessary

Cons:

- Many concepts are unklar definiert, which leads to problems with tool-integration



Systems Modeling Language Version 2 (SysML v2)

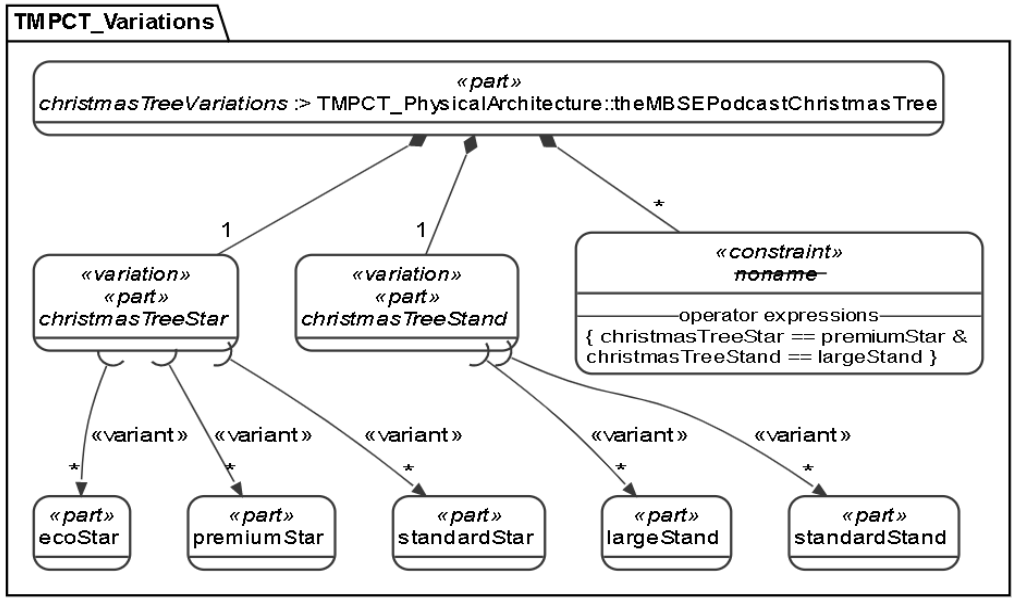
SysML v2 is a significantly improved successor to **SysML v1**. It breaks away from its UML foundation, enabling a better performance in multiple ways.

Improvements over v1:

- Precision and expressiveness of the language
- Consistency across diagrams
- Improved tool and model integration
- Easier to understand
- More flexible for specific domains

Cons:

- Due to the language being released quite recently there are no tool options for our Use Case, that support the creation of SysML v2 diagrams



SysML v2 is the best modeling language for the system in our use case. However, due to the lack of adequate tool support, it was not possible to select this language. Unfortunately, the same applies to SysML v1.

Modeling Tool Evaluation

Several factors influenced our **decision** when choosing between **tools**. The **key factors** were:

- The tool must be open-source or free of charge
- It shoud be usable across all major operating systems
- It must support XMI export for compatibility with traceability tools
- It should be accesible and easy to use

Evaluated Tools for SysML v1:

- Cameo Systems Modeler
- Papyrus for SysML
- Modelio
- SysML v2 Pilot Implementation
- Enterprise Architect
- Umple
- PlantUML

All tools except PlantUML **failed to meet multiple points of the key criteria**. Therefore, **PlantUML was chosen despite** its **incompatibility** with any **traceability tools**. Throughout the evaluation process, no alternative was found to better meet our software requirements.
Therefore, the final decision was to **adopt UML as the modeling language for the future**.

PlantUML: Benefits and How to Address Its Limitations

Among all the software evaluated, PlantUML stands out not only as an open-source solution but also as by far the easiest to integrate into the standard development process, as well as the most intuitive tool for creating models.

Solutions for the problem maintaining Traceability:

Introduced Model-Name-Standard:

- Use Case Diagram: M_US1_UCD1.1.puml
- Class Diagram: M_US2+5_CSD3.1.puml
- Sequence Diagram: M_US5-7_SQD1.0.puml
- Component Diagram: M_US4_CPD2.2.puml
- Deployment Diagram: M_US6_DPD1.1.puml

In addition to the .puml file, the model can also be stored as a .png.

On the next slide, a final attempt will be made to reduce the effort required to maintain traceability between models.

Architecture Decision Records (ADR's):

- ADR's are created for every single model and contain valuable information about them in a short version
- They can also be used to document both **upstream** and **downstream traceability**
- ADR's are markdownfiles and should have the same name as their respective models:
  Ex.: M_US1_UCD1.1.md
- It is is necessary to follow and update the Up-/ and Downstream of a model if it was updated.

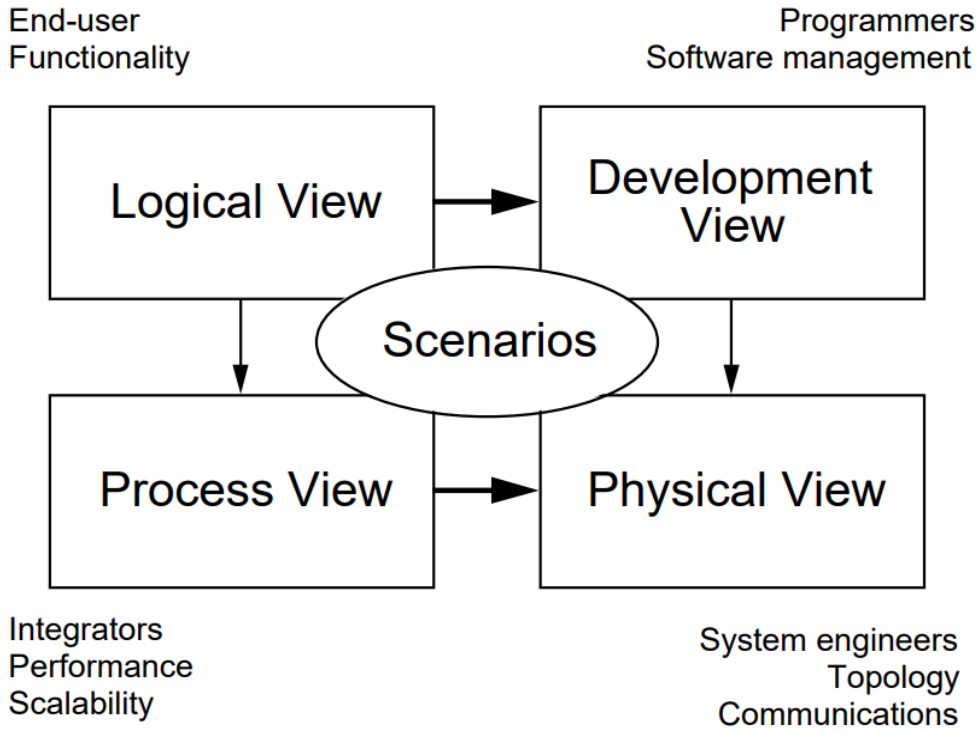
UML: Downsizing of its Generality

As stated previously a **challenge** with UML is, that its **generality can lead to unnecessary complexity**. To adress this we decided to use the **"4+1 View Model"**. By doing so we can concentrate on the most important views for a comprehensive perspective of our System.

"4+1" View Model: Realization in UML

- **Scenarios:** Use Case Diagram
- **Logical View**: Class Diagram
- **Process View**: Sequence Diagram
- **Development View**: Component Diagram
- **Physical View**: Deployment Diagram

This model enables us to use UML efficiently and therefore reduces the workload required to ensure traceability with PlantUML.



Kruchten, Philippe. "Architectural blueprints-the" 4+ 1" view model of software architecture.", Figure 1, *IEEE software* 12.6 (1995): 42-50

Referenzen: Vorgestellte Modelliersprachen

- Delsing, J., Kulcsár, G. & Haugen, Ø. SysML modeling of service-oriented system-of-systems. *Innovations Syst Softw Eng* **20**, 269–285 (2024). https://doi.org/10.1007/s11334-022-00455-5
- https://innovationspace.ansys.com/product/introduction-to-sysml-v2/
- https://www.omg.org/cgi-bin/doc?syseng/25-03-04.pdf
- Kahlaoui, Abdelilah, Alain Abran, and E. Lefebvre. "Dsml success factors and their assessment criteria." *Metrics News* 13.1 (2008): 43-51.
- A. Falcone, A. Garro, A. D'Ambrogio and A. Giglio, "Using BPMN and HLA for SoS engineering: lessons learned and future directions," 2018 IEEE International Systems Engineering Symposium (ISSE), Rome, Italy, 2018, pp. 1-8, doi: 10.1109/SysEng.2018.8544399.
- https://www.bpmhandbook.com/volume-1/table-of-content/business-process-model-and-notation-bpmn/
- P. B. Kruchten, "The 4+1 View Model of architecture," in *IEEE Software*, vol. 12, no. 6, pp. 42-50, Nov. 1995, doi: 10.1109/52.469759.

Referenzen: Evaluierte Software

- https://plantuml.com/de/
- https://cruise.umple.org/umple/
- https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation
- https://www.3ds.com/products/catia/no-magic/cameo-systems-modeler
- https://eclipse.dev/papyrus/
- https://www.modelio.org/index.htm
- https://www.sparxsystems.de/