

Progress Report: Development of a Tool for Boxology

Mahsa Forghani Tehrani

April 2025

1 Agenda

- Introduction
- Review of Four Selected Papers
- Design Pattern in DOT Language
- Design Pattern based on Draw.io
- Boxology Language

2 Introduction

This report presents the progress made in the development of a tool for Boxology—a visual language used to describe and analyze hybrid AI systems. The goal of this work is to explore existing design pattern frameworks and create a structured tool that supports the modeling, validation, and documentation of these systems. As a first step, relevant literature has been reviewed to understand the foundations of Boxology, followed by initial efforts to implement visual patterns using Graphviz and Draw.io. The outcomes presented here reflect the foundational work done so far and provide a basis for further development in the next phases.

3 Review of Four Papers

3.1 A Boxology of Design Patterns for Hybrid AI Systems

[HT19]

This paper presents building blocks that define different parts of systems figure 1: input, output, and process. It categorizes systems into two families:

- **Task-based distinction:** Deductive inference (KR), Inductive learning (ML)

- **Representation-based distinction**

3.1.1 Design Patterns in Software Engineering

[Gam+94]

- Four essentials to solve a problem: pattern name, problem, solution, and consequences.

3.1.2 Design Patterns in Knowledge Engineering

[Sch+94]

- CommonKADS (Common Knowledge Acquisition and Design System)
- Links between knowledge models and actual code.

The main focus is on covering different components and relationships in systems based on a large body of literature, treated as building blocks.



Figure 1: Building blocks

3.2 Modular Design Pattern for Hybrid Learning and Reasoning Systems

[Bek+21]

This paper builds upon the earlier Boxology work and introduces elementary patterns. These patterns can be extended and combined to support more complex systems that integrate both data-driven and knowledge-driven AI components.

3.2.1 Design Patterns in Software Engineering

[Gam+94]

The main contribution is the extension of modular boxology into a simplified framework that supports the development of hybrid learning and reasoning systems.

3.3 Combining Machine Learning and Semantic Web

[Bre+23]

This paper evaluates architectural and application-specific characteristics. It proposes a study protocol to survey existing systems in order to determine how,

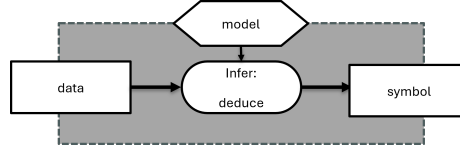


Figure 2: Elementary pattern 2a

when, and in which contexts SWeMLS (Semantic Web Machine Learning Systems) are used. Systems are classified based on their ML and SW components and their interactions.

3.3.1 Key Contributions

- Visualization and notation-based representations of systems across various domains.
- Meta-processing flow of interaction patterns.
- Formalized classification of SWeML systems as an ontology: *SWeMLS Ontology* [Eka+22].

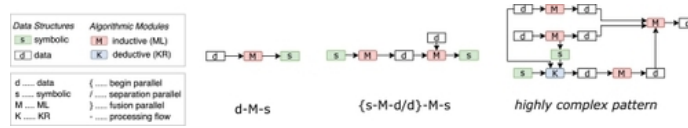


Figure 3: Visual (top) and flat (bottom) notation of system patterns according to the SWeML Systems Boxology.

3.4 Pattern-Based Engineering of Neurosymbolic AI Systems

[Eka24]

This paper supports the engineering process of Neurosymbolic AI systems across their entire life cycle, using design patterns and knowledge graphs.

3.4.1 Prior Works on Neurosymbolic AI Systems

- Machine-actionable representation of SWeML systems: SWeML ontology.
- Visual editor for AI system representation: BEAM, which converts visual structures into a JSON-based format.
- Advanced applications based on system representation: exploratory search, advanced analysis of NeSy-AI system data, and support for AI system auditability.

3.4.2 Pattern-Based AI System Engineering

- **A:** Formalization, extraction, and visualization of AI system representations.
- **B:** Engineering and documentation of AI systems using design patterns.

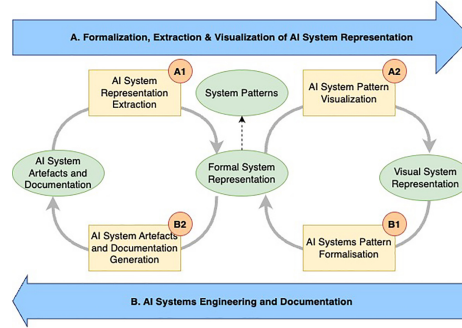


Figure 4: A vision of pattern-based AI systems engineering.

3.5 Comparison Based on Represented Design Patterns

Aspect	Boxology of Design Patterns for Hybrid Systems	Modular Design Patterns for Hybrid Learning	Combining Machine Learning and Semantic Web
Focus	Based on the task performed by the system	Based on the task of the underlying system	Separate from the underlying system
Taxonomy	Uses a structured taxonomy to describe hybrid systems	Introduces taxonomies for processes, data structures, and actors	Provides a classification system for SWeML systems
Design Patterns	Boxology	Boxology	Boxology Textual annotation Ontology
Total Patterns	15	16	41

Table 1: Comparison of Design Patterns

3.6 AI System Representation Toolkit:BEAM

[Gro25]

This toolkit provides a notation library for system modeling in Draw.io. The extended version focuses on:

- Transparency
- A balance between flexibility and ease of learning
- Scalability
- Modularity
- Granularity
- Usability for conversion, reputability, and querying

3.7 Comparison of Visual Representations

Visualization of hybrid AI systems plays a crucial role in clarifying their structural components and operational flows. To effectively represent different processes and modules, a complete and consistent visual vocabulary is necessary.

In this step, I recreated the Skill-Matching case study from [Bek+21] using the BEAM Toolkit [Gro25]. The resulting diagram highlights certain limitations of the BEAM framework in serving as a unified representation for hybrid AI systems. As shown in the figures below, the Boxology visualization (Figure 5) provides a more modular and semantically rich depiction compared to the more abstract and simplified visualization produced with BEAM (Figure 6).

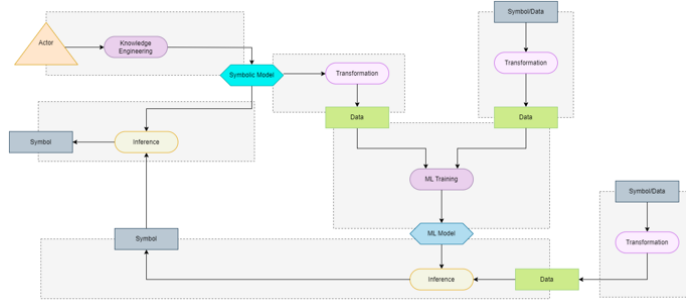


Figure 5: Skill-Matching use case presented in Boxology notation.

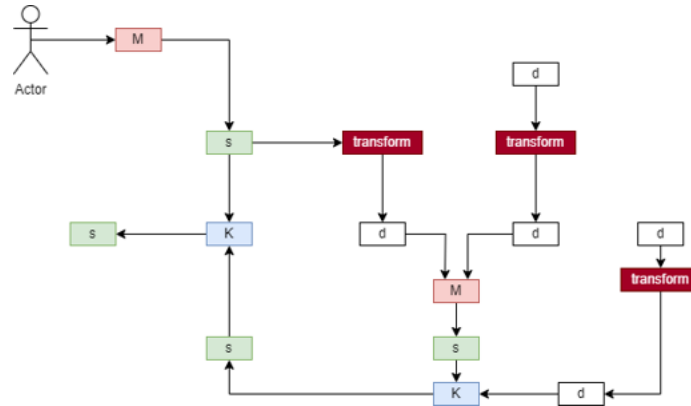


Figure 6: Skill-Matching use case using BEAM representation (SWeML-based).

4 Design Pattern in DOT Language

In this step, the goal is to visualize the elementary pattern described in [Bek+21]. For this purpose, we use the Graphviz Visual Editor, which allows the development of visual structures using the DOT language. While DOT enables basic visualization and shape creation, it lacks the expressive power needed to support our objective—developing a unified language for Hybrid AI systems with a rich vocabulary and grammar.

You can view the elementary pattern as represented in DOT format in this reference [SDM25b].

5 Design Patterns Based on Draw.io

Draw.io (also known as diagrams.net) is a free, open-source diagramming tool that can be used directly in a web browser or as a desktop application. It offers a wide range of shapes for constructing various types of diagrams and

supports the development of JavaScript-based plugins for custom functionality. This makes it an ideal platform for defining a visual vocabulary for the Boxology language, where shapes represent concepts (vocabulary) and plugin logic enforces structural rules (grammar).

The plugin and corresponding vocabulary can be found in the following reference: [SDM25a].

5.1 Vocabulary Library

Based on [Bek+21], I have defined shapes as vocabulary elements in this library. Each shape is uniquely labeled, so using an undefined or incorrect shape will result in validation failure. This library also includes a set of predefined, validated elementary design patterns that can be inserted into the canvas, combined with other patterns, and checked for consistency.

5.2 Boxology Validation

This is a JavaScript-based plugin that can be executed within the Draw.io application or directly in the web browser console. The Boxology validation plugin enforces structural rules between shapes to constrain their combinations and ensure conformity with the defined elementary design patterns.

To implement these constraints, we consider different aspects of validation:

5.2.1 Next Valid

The sequence of shape connections must be valid. For example, connecting a **data** shape directly to a **symbol** is semantically incorrect. This validation step helps users avoid invalid connections both within a single pattern and between multiple patterns.

5.2.2 Valid Pattern

Valid patterns are those defined in [Bek+21]. Even if individual components are connected in a valid sequence, the overall structure must still conform to one of these elementary design patterns. This ensures high-level semantic correctness.

5.2.3 Merging

In complex systems with interconnected subsystems, two systems can only be linked via a shared connector component. Otherwise, an intermediary process must be introduced to reflect their contribution. When two identical components are connected, they are automatically merged to avoid duplication and clarify data/control flow.

1. Grammar Symbols

Language to Specify the Production Rules

- $\{A;B;C\}$:= Represents multiple (A B C or more) inputs or outputs.

Terminal Symbols (T):

- $\langle \text{data_symbol} \rangle$:= Represents data or symbols.
- $\langle \text{model} \rangle$:= Represents a statistical or semantic model.
- $\langle \text{actor} \rangle$:= Represents an autonomous agent (human, software, robot).
- $\langle \text{generate_train} \rangle$:= Represents a process that trains a model.
- $\langle \text{generate_engineer} \rangle$:= Represents a process where an actor engineers a model.
- $\langle \text{transform} \rangle$:= Represents a transformation process that modifies data.
- $\langle \text{infer_symbol} \rangle$:= Represents an inference process that derives a new symbol.
- $\langle \text{infer_model} \rangle$:= Represents an inference process that derives a new model.
- $\langle \text{infer_deduce} \rangle$:= Represents an inference process that derives new data or models.
- $\langle \text{left_right_arrow} \rangle$:= Represents an arrow from left to right.
- $\langle \text{arrow} \rangle$:= Represents an arrow.
- $\langle \text{title} \rangle$:= Represents the title of a pattern.

Non-Terminal Symbols (N):

- $\langle \text{elementary_pattern} \rangle$:= Represents an elementary pattern in the boxology.
- $\langle \text{elem_pattern_generate_model} \rangle$:= Represents an elementary pattern to generate a model.
- $\langle \text{elem_pattern_use_model} \rangle$:= Represents an elementary pattern to use the model.

Production Rules (P)

```
<elementary_pattern>      := <elem_pattern_generate_model> | <elem_pattern_use_model>
<elem_pattern_generate_model>:= <train_model> | <create_model> | <generate_model>
<train_model>             := <data_symbol> <left_right_arrow> <generate_train>
<left_right_arrow> <model>
<create_model>            := <actor> <left_right_arrow> <generate_engineer>
<left_right_arrow> <model>
<generate_model>          := <data_symbol> <left_right_arrow> <transform>
<left_right_arrow> <data_symbol>
<elem_pattern_use_model>   := <infer_symbol> | <infer_model>
<infer_symbol>            := {<data_symbol> <left_right_arrow>; <model> <top_down_arrow>}
<infer_deduce> <left_right_arrow> <symbol>
<infer_model>             := {<data_symbol> <left_right_arrow>; <model> <top_down_arrow>}
<infer_deduce> <left_right_arrow> <model>
```

Extended Grammar for Nested and Combined Patterns

Extra Non-Terminal Symbols

- <complex_pattern> := Represents a combination of multiple elementary patterns.
- <diagram> := Represents a full boxology diagram with multiple complex patterns.

Extended Production Rules

```
<complex_pattern> ::= <elementary_pattern>
                  | <complex_pattern> <left_right_arrow> <complex_pattern>
                  | {<complex_input>} <complex_pattern> {<complex_output>}
<complex_input>  ::= <complex_pattern> <arrow> | <complex_input>; <complex_input>
<complex_output> ::= <arrow> <complex_pattern> | <complex_output>; <complex_output>
<diagram>       ::= <title_pattern> <complex_pattern> | <diagram> <diagram>
```

References

- [Gam+94] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1994. ISBN: 978-0-201-63361-0. URL: <https://books.google.de/books?id=6oHuKQe3TjQC>.
- [Sch+94] Guus Schreiber et al. “CommonKADS: A comprehensive methodology for KBS development”. In: *IEEE Expert* 9.6 (1994), pp. 28–37. DOI: 10.1109/64.363263. URL: https://www.researchgate.net/publication/220628697_CommonKADS_A_comprehensive_methodology_for_KBS_development.
- [HT19] Frank van Harmelen and Annette ten Teije. “A Boxology of Design Patterns for Hybrid Learning and Reasoning Systems”. In: *arXiv preprint arXiv:1905.12389* (2019). URL: <https://arxiv.org/abs/1905.12389>.
- [Bek+21] Michael van Bekkum et al. *Modular Design Patterns for Hybrid Learning and Reasoning Systems: A Taxonomy, Patterns and Use Cases*. Preprint under review. 2021. arXiv: 2102.11965 [cs.AI]. URL: <https://arxiv.org/abs/2102.11965>.
- [Eka+22] Fajar J. Ekaputra et al. *Semantic-Web Machine Learning System (SWeMLS) Ontology*. Version 1.0. Dec. 2022. URL: <https://semsys.ai.wu.ac.at/ns/swemls/>.
- [Bre+23] Anna Breit et al. “Combining Machine Learning and Semantic Web: A Systematic Mapping Study”. In: *ACM Computing Surveys* 55.14s (2023), Article 313. DOI: 10.1145/3586163. URL: <https://doi.org/10.1145/3586163>.
- [Eka24] Fajar J. Ekaputra. “Pattern-Based Engineering of Neurosymbolic AI Systems”. In: *Journal of Web Semantics* (2024). Manuscript draft, JOWS-D-24-00080. URL: <https://www2.cloud.editorialmanager.com/jows/viewRCResults.aspx?pdf=1&docID=6006&rev=0&fileID=86879&msid=73fe9e06-ce78-4af5-9704-8449fcb731fc>.
- [Gro25] WU SemSys Group. *BEAMv3 Instructions*. Accessed: 2025-03-31. 2025. URL: https://github.com/wu-semsys/beam_tutorial/blob/main/BEAMv3-instructions.pdf.
- [SDM25a] SDM-TIB. *Tool4Boxology: BoxologyPlugin*. <https://github.com/SDM-TIB/Tool4Boxology/tree/main/BoxologyPlugin>. Accessed: 2025-03-31. 2025.
- [SDM25b] SDM-TIB. *Tool4Boxology: Elementary Pattern*. <https://github.com/SDM-TIB/Tool4Boxology/tree/main/ElementaryPattern>. Accessed: 2025-03-31. 2025.