MONALISA

Monitoring and Analytics for the whole Lifecycle, on Models, Hardware, and Software

D4.1 Specifications and Designs of the Computational Models

Deliverable Due Date	2026-01-16
Deliverable Type	Document
Deliverable Number	D4.1
Authors	László Tóth (University of Szeged)
Dissemination Level	Public

Contributors

Name	Short Affiliation
László Tóth	University of Szeged (Hungary)

Reviewers

Name	Short Affiliation	Date
Matthias Mohlin	HCL	2026.01.14

Document Revision Log

Version	Revision	Date	Description	Author
1	01	2025-12-09	First Draft	László Tóth (USZ)
1	02	2025-12-14	Revision	László Tóth (USZ)
1	03	2025-12-16	Final Version	László Tóth (USZ)
1	04	2026-01-16	Submitted	László Tóth (USZ)

Table of Contents

List of Figures

List of Tables

Definitions and Acronyms

- **MONALISA** - Monitoring & Analytics for the Whole Lifecycle, from Concept to System Models, Hardware & Software
- **AI** - Artificial Intelligence
- **API** - Application Programming Interface
- **CPS** - Cyber-Physical Systems
- **UML** - Unified Modeling Language
- **UML-RT** - UML for Real Time
- **SysML** - System Modeling Language
- **XML** - eXtensible Markup Language
- **DSML** - Domain-Specific Modeling Language
- **MoC** - Model of Computation
- **LLM** - Large Language Model
- **GPT** - Generative Pre-trained Transformer
- **JANI** - JSON-based intermediate modeling language
- **PNML** - Petri Net Markup Language
- **XMI** - XML Metadata Interchange
- **SCXML** - State Chart XML

1. Executive Summary of Deliverable

This delivery provides an in-depth analysis of WP4 (model reengineering). Further, it is the base of the implementation of WP4. The objective of this executive section is to inform of the project's results, highlight significant findings, and recommend actionable steps moving forward. Overall, the analysis shows the different capabilities of LLM-based approaches vs. the use of mathematical models, which suggests combining the strengths of both techniques. Further, the summary defines the context and interfaces with which WP4 interacts with other work packages.

This deliverable specifies the computational modeling framework developed in Work Package 4 (WP4) of the MONALISA project. Its primary objective is to define a unified, formally grounded approach for representing, extracting, and analyzing system behavior across the full lifecycle—from natural-language requirements and design documents to runtime execution traces.

WP4 addresses a key industrial challenge: ensuring consistency between intended system behavior described in informal documentation and realized behavior observed in execution logs. To this end, the work combines formal Models of Computation (MoCs) with large language model (LLM)–assisted extraction techniques, enabling automated translation from both textual and trace-based inputs into analyzable formal models.

The deliverable positions WP4 as a central integration layer within the MONALISA analysis pipeline. Harmonized trace data provided by WP3 and conceptual specifications from multiple sources are ingested by WP4, transformed into formal models, and then passed to WP5 for verification, semantic analysis, and root-cause evaluation. This establishes end-to-end traceability from requirements to execution-level evidence.

A limited but expressive set of MoCs is selected based on industrial relevance, toolchain maturity, and suitability for automated generation. These include UML and SysML models for structural, behavioral, and process-level representation, as well as Petri Nets and Timed Automata for detailed concurrency and real-time analysis. Each MoC is defined with precise syntax and semantics and mapped to standardized interchange formats to ensure interoperability and downstream tool support.

The methodological framework integrates three key elements:

(1) LLM-based natural-language formalization, which extracts structured models from informal requirements and specifications;
(2) trace-based model inference, which reconstructs behavioral models from execution traces; and
(3) formal metamodel definition, providing a shared semantic foundation across modeling formalisms.

To support implementation and integration, the deliverable defines a common core metamodel and a set of deterministic APIs for trace ingestion, model generation, behavioral inference, and formal verification. These interfaces enable reproducible transformations and seamless interaction between modeling and analysis components.

In conclusion, this work establishes a scalable and coherent modeling foundation for MONALISA, enabling systematic comparison of conceptual designs and observed system behavior. The defined framework supports advanced analysis tasks, including conformance checking, anomaly detection, and timed property verification. Next steps include finalizing model profiles per use case, implementing the defined services, and piloting end-to-end workflows across selected industrial scenarios.

2. Introduction

This deliverable provides an in-depth analysis of the **specification and design of unified computational models** developed in **Work Package 4 (WP4)** of the *Mona Lisa* project.

The objective of this section is to inform the consortium of the project's results, highlight significant conceptual findings, and outline actionable next steps for model integration and validation.

Overall, the analysis demonstrates that **formalized models of computation (MoCs)**—supported by **large language models (LLMs) — based translation from natural-language and trace-based inputs** can serve as a coherent foundation for analysis, verification, and cross-domain interoperability within the project. This approach enables consistency checking between conceptual system descriptions and their realized behaviours captured in trace logs.

This summary defines the context of WP4 in relation to other tasks and work packages:

- **WP3** provides data in **Google Trace Format and JSON,** along with the relevant data interfaces, serving as the primary inputs to this task.
- **WP4**, through Task 4.1, specifies the unified metamodels, syntax, and semantics for a selected set of Model of Computation (MoC) models.
- **WP5** will consume these models for **analysis, verification, and root-cause evaluation**.

The methodology combines **formal modeling**, **metamodel definition**, and **machine-learning-assisted model extraction**, enabling industrial use cases to be represented and analyzed within a harmonized computational framework.

2.1 Objectives of the WP4

The overarching goal of WP4 is to establish a unified framework for representing and analyzing computational behavior across different application domains. By leveraging formal models of computation, WP4 enables a systematic comparison of conceptual system designs and their real-world implementations, thereby supporting the identification of discrepancies, root causes of design deviations, and optimization opportunities.

2.1.1 Specific objectives of Task 4.1 – Model Definitions and Platform

Task 4.1 focuses on defining a coherent set of models of computation that capture the relevant aspects of all project use cases. The task pursues the following objectives:

- Identify and formalize a limited but expressive set of MoCs that can represent the dynamics of various industrial systems.
- Define unified data structures and programming interfaces to ensure interoperability among WP3 (Trace Data), WP4 (Computation Models), and WP5 (Analysis and Verification).

- Establish the formal syntax and semantics of each selected MoC.
- Apply natural language processing and LLM-based methods to extract and align formal models from textual specifications.
- Defining algorithms to extract a higher-level representation of trace logs.
- Create a foundation for subsequent analysis tasks, including anomaly detection, correlation analysis, timed property verification, and design-space exploration.

The relationships between the work packages are presented in Figure 1.

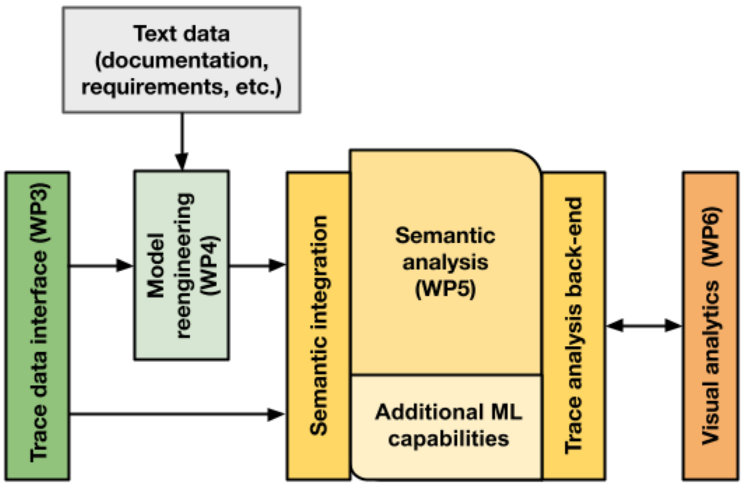
Text data (documentation, requirements, etc.)
Trace data interface (WP3)
Model reengineering (WP4)
Semantic integration
Semantic analysis (WP5)
Additional ML capabilities
Trace analysis back-end
Visual analytics (WP6)

Figure 1. ***Interaction between the analysis work packages and their role within the overall pipeline.***

3. Input Sources and Data Flow

The inputs to WP4 originate from three primary sources:

1. **Conceptual, design, and requirements documents** are natural-language descriptions of system requirements, design specifications, and operational scenarios. LLM-based transformation pipelines will translate these textual descriptions into structured, formal models in accordance with predefined MoCs. These sources typically do not exist as simple text files in a single location; they may be scattered across several systems and tools, such as Jira, PM tools, Word documents, and Wiki pages.

2. **Test cases and test logs** are typically stored in semi-structured documents that can be processed by an LLM.

3. **Trace Logs and Runtime Data** – Actual system execution traces are collected by industrial partners and harmonized by WP3 into a Google Trace Format [6] and JSON. These standardized trace files serve as the empirical foundation for deriving behavioral models that represent realized system behavior.

The relationship between the sources is illustrated in the following data flow:

- WP3 provides harmonized trace data via standardized interfaces.
- WP4 ingests both textual and trace inputs.
- LLMs are applied to generate formal representations in MoCs.
- Suitable algorithms generate a higher-level representation of the trace file.
- The resulting models are then analyzed and compared to detect inconsistencies, performance bottlenecks, or design flaws.
- WP5 uses these models for deeper analysis and verification and provides the results to WP6.

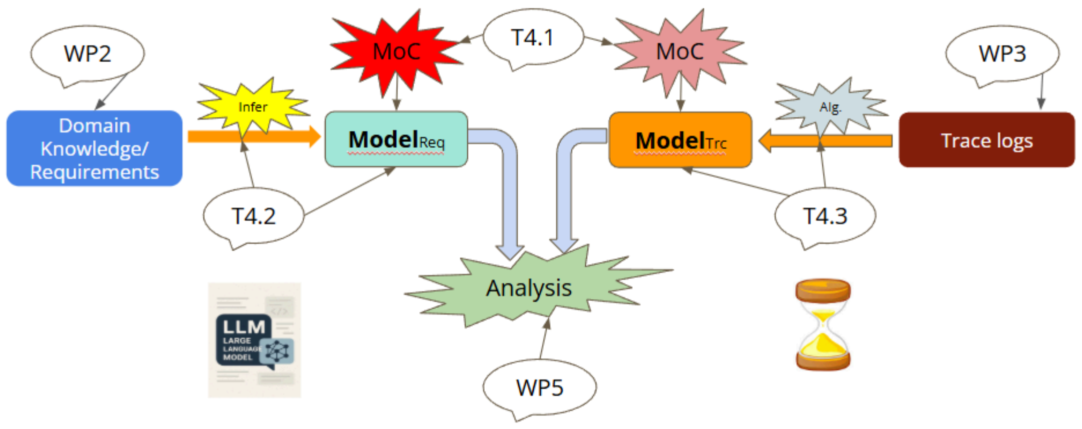

Figure 2. The Data Flow from WP2/WP3 to WP5

4. Methodology and Model Formalization Approach

The methodological framework of WP4 combines **formal modeling, model transformation, and machine learning components** into a unified workflow.

4.1 Natural Language Processing and Formalization

LLMs (such as GPT-based architectures) are used to interpret informal requirements and transform them into well-structured models compliant with selected MoCs. The workflow involves:

- **Information extraction** from textual sources using domain-specific prompts and ontologies;
- **Mapping** of linguistic constructs (requirements, constraints, relationships) to model elements (states, transitions, parameters);
- **Generation of metamodel-compliant outputs** (e.g., UML/SysML XMI files or Petri Net structures).

LLM can serve as a semantic bridge between natural language and formal representation.

4.2 Trace-Based Model Extraction

For behavioral analysis, WP4 processes trace logs in the standard format delivered by WP3. These traces are transformed into executable or analyzable models. The applied model-specific algorithms assist with pattern recognition, event classification, and the identification of synchronization and causal relationships between observed events.

4.3 Formalization and Metamodel Definition

Each selected MoC is formally defined by:

- A **metamodel** specifying its elements, relationships, and constraints;
- A d**omain-specific syntax,** expressed either as a DSL or ontology;
- A **mathematical semantic**s, providing precise meaning to computational constructs.

This document introduces the structure and rationale for these MoCs and provides the foundation for implementation in later deliverables.

5. Specifications of the Models of Computation

5.1 Rationale and Selection Principles

Model extraction from specifications, designs, and test cases uses LLMs because these documents are written in natural language. The inferred models depend on the needs of the use case providers. The table in the following subsection lists the inferred models for each use-case provider.

Similarly, model inference from trace logs is use-case-specific. The table also lists the models inferred from trace logs for each use case.

The selection balances:

- **Faithfulness** to industrial processes (concurrency, resources, timing),
- **LLM-friendliness** (stable syntax, taxonomy, and metamodel targets),
- **Toolchain maturity** (existing verifiers/simulators),
- **Comparability** (ability to align conceptual vs. realized models).

5.2 Overview of the Applied Models

For both trace logs and requirements documentation, a wide range of models is available. In engineering practice, SysML [2] and UML [1] are widely used; however, several other modeling languages are also suitable for process management. Given their widespread adoption and graphical representability, this project focuses on SysML/UML models; however, in some cases, more detailed models, such as timed automata [5] or Petri Nets [3], may also be used based on the needs of the use case providers. These models are supported by multiple tools, so diagram rendering during visual analysis and output interpretation is straightforward.

The selected models are as follows:

Input source (Document or trace log)	Recommended MoC(s)	Typical diagrams/forms	Primary target	Fit for industrial processes	Key analyses enabled
Trace log	Message Sequence Charts / UML Sequence	Lifelines, messages, combined fragments	XMI	Excellent (interfaces, timing approximations)	Ordering/ latency conformance, protocol check
	Timed automata	Clocks, invariants, guards	JANI [4]	Excellent (real-time behavior)	Timed property verification, schedulability
	Petri Net	Places, transitions, tokens	PNML [3]	Excellent (concurrency, buffers)	Ordering/ latency conformance, protocol check
Requirements & Conceptual docs	UML Use Case & Activity	Use Case; Activity (control/data flows)	XMI	Excellent (process narratives)	Flow completeness, pre/post-condition consistency
	UML/SysML State Machine	Statecharts with orthogonal regions	XMI / SCXML [7]	Excellent (mode logic, interlocks)	Reachability, deadlock absence, and guarding conditions

Table 1: Selected Mocs

5.3 MoCs Description

5.3.1 UML Sequence / Message Sequence Charts (MSC)

- **Fit**. Interface-level causal order, request/response patterns, and retry schemes.
- **From Google Trace Format.** Map event streams to lifelines (components), messages (event pairs), and combined fragments (alt/loop/par).
- **Serialization**. XMI.
- **Analyses**. Protocol conformance, latency budgets, and timeout detection.

5.3.2 UML Use Case & Activity

- **Use Case.** Captures actors, goals, and interactions; good for early conformance anchors.
- **Activity**. Encodes control/data flows, decision/merge, fork/join; ideal for process structure and exception paths.
- **Serialization**. XMI.
- **LLM mapping.** Paragraph → Actor/Action/Object/Guard tuples → Activities with partitions (swimlanes).

5.3.3 UML/SysML State Machine

- **Scope**. Modes, interlocks, error handling, concurrent regions (orthogonal components).
- **Serialization**. XMI (tooling) and SCXML [7] (execution/monitoring).
- **LLM mapping**. Conditional phrases → guards; temporal adverbs → timers/timeouts; failure phrases → error states.
- **Analyses**. Reachability, deadlock/safe state checks; alignment with trace-derived automata.

5.3.4 SysML Structure + Parametric

- Block Definition/Internal Block for structure and interfaces; Parametric for constraints (e.g., throughput = f(buffer, takt, uptime)).
- **Serialization**. XMI.
- **Analyses**. What-if and sensitivity (hooks for WP5 quantitative studies).

5.3.4 Timed automata

- **Fit**. Real-time control (PLC/embedded), sequencing with hard timing constraints.
- **From Google Trace Format.** Extract states by trace clustering; infer guards/invariants from inter-event times; model timeouts as clock constraints.
- **Serialization**. JANI [4] (neutral) or a TA-specific interchange.
- **Analyses**. Timed property verification, schedulability, and deadline misses.

5.3.5 Petri Net

- **Fit**. Industrial processes with buffers, shared resources, concurrency, and blocking.
- **From Google Trace Format.** Activity labels → transitions; resources/buffers → places; token counts from WIP/queue states; timestamps → Timed PN.
- **Serialization**. PNML.[3]
- **Analyses**. Liveness, boundedness, throughput, bottlenecks, and conformance checking (align traces to PN).

5.4 Metamodels and Programming Interfaces (WP4–WP3–WP5)

5.4.1 Core Metamodel (common layer)

Minimal shared vocabulary across MoCs to aid LLMs and adapters:

```
Entity: {id, name, type, attrs}

Relation: {source, target, kind, attrs}

Event: {id, timestamp, actor, label, payload}

Constraint: {scope, expr, kind: [temporal|resource|safety|prob]}

Metric: {name, scope, definition}

TraceRef: {event_id ↔ model_element}
```

5.4.2 Specific Metamodel Hooks

- **UML/SysML (XMI):** profiles for industrial domains; stereotypes for timing/criticality.
- **SCXML:** runtime hooks for state observers (monitoring).
- **Petri (PNML):** place/transition annotations (resource, buffer size, cost).
- **Timed Automata (JANI):** clocks, invariants, guards; property specs (TCTL).

6. API description

6.1 POST /predict_row

Purpose

Single log row anomaly prediction with optional debug metadata.

Request body (JSON)

```
{

"timestamp": "2026-01-08T10:00:00",

"level": "ERROR",

"source": "orders-api",

"message": "Unhandled exception..."

}
```

Response (JSON)

```
{

"is_anomaly": true,

"explanation": "<LLM explanation in Hungarian>",

"metadatas": { ... }   // empty object if debug is OFF

}
```

Behaviour

- Converts the request into an internal row dict and calls check_log_anomaly.
- `If TOGGLE_DEBUG == 0` (debug OFF): returns only is_anomaly and explanation, with metadatas as an empty {}.
- `If TOGGLE_DEBUG == 1` (debug ON): check_log_anomaly returns (is_anom, explanation, metadata) and the metadata field is populated with model-related metadata (e.g., retrieved documents, scores, etc.).

6.2 `POST /predict_file`

Purpose

Batch anomaly prediction over an uploaded CSV file.

Content type: `multipart/form-data`

Request: upload a file to the `file` field.

Expected CSV columns:

- `timestamp`
- `level`
- `source`
- `message`

Behavior:

- Reads the uploaded CSV into a pandas DataFrame.
- Runs `predict_dataframe(df)` to apply anomaly detection row by row.
- `predict_dataframe` appends at least two new columns to each row:
  - `anomaly` (boolean)
  - `explanation` (string, LLM explanation)

Response:

- Returns a CSV file (`text/csv`) as an attachment via `StreamingResponse`.
- The filename is prefixed with the currently selected LLM model name:
  `predicted_<CURRENT_MODEL_NAME>_<original_filename>.csv`.

6.3 POST /set_model

Purpose

Set the default Ollama LLM model used by the API.

Query parameter:

- `model` (Enum `LLMModel`): selectable list of available Ollama models (e.g. `llama3.1`, `mistral`, etc.), exposed as a drop-down in Swagger.

Behavior:

- Updates the global `CURRENT_MODEL_NAME` to the selected value.
- Re-creates the global `llm` instance via `get_llm_for_current_model()`, so subsequent calls to `/predict_row` and `/predict_file` use the newly selected model.

Response (JSON):

```
{

"current_model": "llama3.1",

"message": "LLM model set to llama3.1"

}
```

6.4 POST /set_embedding

Purpose

Set the default embedding model and associated Chroma collection.

Query parameter:

- `em_model` (Enum `EmbeddingModel`): selectable list of embedding models (e.g. `nomic-embed-text`, etc.), shown as a drop-down in Swagger.

Behavior:

- Updates `CURRENT_EMBEDDING_NAME` to the selected embedding model name.
- Re-creates the global `embeddings` instance via `get_embedding_for_current_model()`.
- Re-initializes the global `vectordb` as:

```
vectordb = Chroma(
       `persist_directory="./log_rag_db",`
       `collection_name=CURRENT_EMBEDDING_NAME,`
       `embedding_function=embeddings,`
)
```

- This means the active RAG index (Chroma collection) is switched to the one that matches the selected embedding function.

Response (JSON):

```
{
"current_model": "nomic-embed-text",
"message": "Embedding model set to nomic-embed-text"
}
```

6.5 POST /toggle_debug

Purpose:

Enable or disable global debug mode.

Query parameter:

- `debug` (Enum `DebugFlag`):
    - `0` → `OFF` (default)
    - `1` → `ON`

      Exposed as a 0/1 drop-down in Swagger.

Behavior:

- Sets the global `TOGGLE_DEBUG` integer to `0` or `1`.
- When `TOGGLE_DEBUG == 1`, endpoints like `/predict_row` return additional metadata (e.g., retrieved context, raw LLM info), and any debug-only logging you implemented is activated.

Response (JSON):

```
{

"debug": 1,

"message": "Debug flag set to 1"

}
```

6.6 POST /add_csv_to_vectordb

Purpose:

Add CSV rows as documents to the current Chroma collection.

Content type: multipart/form-data

Request: upload a file to the file field.

Expected CSV columns (recommended):

- timestamp
- level
- source
- message

Behavior:

- Reads the uploaded CSV into a pandas DataFrame.
- For each row, build a Document:
    - page_content: a text representation of the log row using row_to_text(row)
    - metadata: a dict containing selected fields (e.g. timestamp, level, source).
- Calls vectordb.add_documents(docs) so that all new documents are embedded with the currently active embedding model and stored in the current Chroma collection (defined by CURRENT_EMBEDDING_NAME)

Response (JSON):

```
{

"added_rows": 123,

"collection_name": "nomic-embed-text",

"message": "Successfully added 123 documents to collection 'nomic-embed-text'.The collection was {was_len} long, now {curr_len} long."

}
```

Usage notes:

- The target collection and embedding function are determined by the last call to /set_embedding.
- After calling /add_csv_to_vectordb, subsequent /predict_row and /predict_file requests will be able to retrieve context from the newly added documents via the shared vectordb.

7. LLM Pipelines: Prompts, Guards, and Validation

- **Schema-constrained generation.** Use JSON/XMI/PNML/JANI **schemas** to force valid structures.
- **Dual extraction.**
    a. Requirements → specs (SysML/UML).
    b. Traces → behavior (PN/TA/Seq).
- **Cross-validation.** Regenerate from model to English summary → compare to source text.
- **Semantic guards.** Ontology terms (actors/resources) must be reused consistently.

Property scaffolding. Auto-draft properties (e.g., “No buffer overflow”, “Deadline ≤ D”) for WP5 verification.

8. Conclusions and Next Steps

This extension provides **actionable guidance** for selecting and applying MoCs to **both** NL requirements and traces. Immediate next steps:

1. Select models applicable to the use cases.
2. Map the different traces to the MoC models.
3. Freeze **XMI/PNML/JANI/SCXML** schema profiles.
4. Implement the reengineering module/service.
5. Implement **trace adapters** and minimal property checks.
6. Pilot two use cases end-to-end (NL→model, trace→model).
7. Integrate WP5 verifiers and produce the **first discrepancy report**.

References

[1] Object Management Group, OMG Unified Modeling Language (OMG UML), Version 2.5.1, formal specification, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1

[2] Object Management Group, OMG Systems Modeling Language (OMG SysML), Version 1.6, formal specification, Dec. 2019. [Online]. Available: https://www.omg.org/spec/SysML/1.6

[3] ISO/IEC, Information technology — High-level Petri nets — Part 2: Transfer format, ISO/IEC 15909-2:2011, International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland, 2011.

[4] S. Budde, J.-P. Katoen, P. Koopman, and M. Stoelinga, “A unified model exchange format for probabilistic model checking tools,” in Proc. 29th Int. Conf. on Computer Aided Verification (CAV), 2017, pp. 71–81.

[5] R. Alur and D. L. Dill, “A theory of timed automata,” Theoretical Computer Science, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[6] Google, Trace Event Format, Chromium Project documentation. [Online]. Available: https://chromium.googlesource.com/catapult/+/HEAD/tracing/docs/trace_event_format.md

[7] J. Barnett, State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Recommendation, Sept. 2015. [Online]. Available: https://www.w3.org/TR/scxml/