



DEPARTMENT OF COMPUTER SCIENCE

VASCO SOARES MESQUITA FERREIRA NUNES

Bachelor in Computer Science

GENERATION OF BUSINESS MODELS IN A LOW-CODE PLATFORM WITH LARGE LANGUAGE MODELS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2026



DEPARTMENT OF COMPUTER SCIENCE

GENERATION OF BUSINESS MODELS IN A LOW-CODE PLATFORM WITH LARGE LANGUAGE MODELS

VASCO SOARES MESQUITA FERREIRA NUNES

Bachelor in Computer Science

Advisers: Vasco Amaral

Assistant Professor, FCT - NOVA University Lisbon

David Aveiro

Assistant Professor, Madeira University

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon

September, 2026

Generation of Business Models in a Low-Code Platform with Large Language Models

Copyright © Vasco Soares Mesquita Ferreira Nunes, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Vasco Amaral, and my co-supervisor, David Aveiro, for their guidance and feedback, and for the time they dedicated to our discussions.

Above all, I thank my family and my girlfriend for their unconditional support, encouragement, and belief in me. None of this would have been possible without them.

ABSTRACT

Low-code platforms allow users to create software through models and visual configuration. DISME is a low-code platform for modelling and executing organisational information systems, based on the Adaptive Object Model (AOM) architecture. However, users still need to understand its modelling concepts to turn business requirements into a system specification. This dissertation aims to facilitate the analysis of these requirements and the derivation of models for DISME through Large Language Models (LLMs), allowing users to describe the intended system in natural language.

The work followed a Design Science Research process. A Systematic Literature Review of 35 studies on LLMs in low-code and model-driven platforms informed the use of structured intermediate representations, automated validation, and human review.

The proposed approach was implemented as a wizard that collects business information through phased interviews and generates a State Model, a Process Model, forms, queries, and Action Rules. Generated artefacts are represented in structured formats, checked by deterministic validators, and presented to the user for review and correction before import into DISME. Action Rules are imported as drafts for inspection and activation in the platform.

An empirical study with 19 participants evaluated the approach using a reduced conference management scenario. All participants completed the accompanied wizard workflow, and almost all considered it useful for producing an initial specification. The tested execution path in DISME was completed in 17 sessions. However, every participant required semantic assistance, particularly when interpreting questions, reviewing Action Rules, and understanding what would be imported into DISME.

The results suggest that the workflow can provide a starting point for users, even if they do not have any DISME experience, to specify systems for an AOM-based platform, while showing where further support is needed.

Keywords: DISME, Large Language Models, Model-Driven Engineering, Low-code platforms, Intermediate representation, Business process modelling

RESUMO

As plataformas low-code permitem criar software através de modelos e configuração visual. O DISME é uma plataforma low-code para modelar e executar sistemas de informação organizacionais, baseada na arquitetura Adaptive Object Model (AOM). No entanto, os utilizadores precisam de compreender os seus conceitos de modelação para transformar requisitos de negócio numa especificação de sistema. Esta dissertação tem como objetivo facilitar a análise destes requisitos e a criação de modelos para o DISME através de modelos de linguagem de grande dimensão (LLMs), permitindo aos utilizadores descrever o sistema pretendido em linguagem natural.

O trabalho seguiu um processo de Design Science Research. Uma Revisão Sistemática da Literatura de 35 estudos sobre LLMs em plataformas low-code e orientadas a modelos orientou a utilização de representações intermédias estruturadas, validação automática e revisão humana.

A abordagem proposta foi implementada num assistente que recolhe informação sobre o negócio através de entrevistas organizadas por fases e gera um Modelo de Estado, um Modelo de Processo, formulários, consultas e Regras de Ação. Os artefactos gerados são representados em formatos estruturados, verificados por validadores determinísticos e apresentados ao utilizador para revisão e correção antes da importação para o DISME. As Regras de Ação são importadas como rascunhos para inspeção e ativação na plataforma.

Um estudo empírico com 19 participantes avaliou a abordagem utilizando um cenário reduzido de gestão de conferências. Todos os participantes concluíram o fluxo de trabalho do assistente com acompanhamento, e quase todos o consideraram útil para produzir uma especificação inicial. O percurso de execução testado no DISME foi concluído em 17 sessões. Contudo, todos os participantes necessitaram de apoio semântico, sobretudo na interpretação das perguntas, na revisão das Regras de Ação e na compreensão do que seria importado para o DISME.

Os resultados sugerem que este fluxo de trabalho pode constituir um ponto de partida para os utilizadores especificarem sistemas para uma plataforma baseada em AOM, mesmo sem experiência prévia com o DISME, indicando também onde é necessário apoio adicional.

Palavras-chave: DISME, Modelos de linguagem de grande dimensão, Engenharia orientada por modelos, Plataformas low-code, Representação intermédia, Modelação de processos de negócio

DISCLOSURE OF USAGE OF GENERATIVE ARTIFICIAL INTELLIGENCE

The author, Vasco Soares Mesquita Ferreira Nunes, acknowledges the use of the following generative artificial intelligence tools in the development of this thesis. These technologies were utilized under strict supervision, and all AI-generated outputs were critically reviewed and verified for accuracy. The author accepts full responsibility for the validity and originality of the content, confirming that the use of these tools aligns with all institutional policies and standards of academic integrity.¹

Generative AI tools used: Sonnet 5 and GPT-5.6 Sol.

Activities supported by Generative Artificial Intelligence Tools:²

Conceptualization

- | | | |
|--|--|---|
| ☐ Idea generation | ☐ Formulating research questions and hypotheses | ☐ Hypothesis viability assessment |
| ☐ Defining the research objective | ☒ Feasibility assessment and risk evaluation | ☒ Simulated debate and argument testing |

Literature Review

- | | | |
|--|--|--|
| ☒ Search and Discovery | ☐ Concept Mapping and Systematization | ☐ Gap Identification and Novelty Evaluation |
| ☒ Literature summarization and synthesis | ☐ Market/patent landscape analysis | ☐ Cross-lingual literature comprehension |

Methodology

- | | | |
|---|--|--|
| ☐ Experimental Design Optimization | ☐ Development of experimental or research protocols | ☐ Methodological Instrument Selection |
|---|--|--|

¹This declaration was generated using pdfL^AT_EX [2] and the L^AT_EX package aidisclose [3].

²This list extends the GAIDeT taxonomy [4].

Software Development and Automation

- | | | |
|--|---|--|
| ☒ Code Generation | ☒ Debugging and Repair | ☐ Algorithm design |
| ☐ Refactoring and Optimization | ☐ Process automation | ☐ Code documentation and comment generation |

Data Management

- | | | |
|---|---|--|
| ☐ Data collection | ☒ Data analysis | ☐ Synthetic data generation |
| ☐ Validation | ☒ Quantitative Plotting and Charting | ☐ De-identification and anonymization support |
| ☐ Data cleaning | ☐ Reproducibility and rerun checks | ☐ Audio-to-text transcription and diarization |
| ☐ Data curation and organization | ☐ Data labeling and annotation assistance | |

Visuals and Multimedia

- | | | |
|---|--|--|
| ☐ Synthetic Asset Generation | ☐ Image Enhancement and Editing | ☐ Diagrammatic and Schematic Design |
|---|--|--|

Writing and Editing

- | | | |
|--|--|--|
| ☐ Drafting Text | ☐ Formulation of conclusions | ☒ Citation formatting and bibliography management |
| ☒ Polishing and Editing | ☒ Tone Adjustment | ☐ Press releases and outreach materials |
| ☐ Abstract and Executive Summary Generation | ☒ Translation | ☐ Title Generation |

Ethics Review

- | | | |
|--|---|--|
| ☐ Bias analysis and discrimination assessment | ☐ Ethical risk analysis | ☐ Data confidentiality monitoring |
| | ☐ Monitoring compliance with ethical standards | |

Quality Assurance

- | | | |
|--|--|--|
| ☒ Simulated Peer Review | ☒ Identification of limitations | ☐ Publication support |
| ☐ Consistency checking against field trends | ☐ Publication strategy and journal selection | |

CONTENTS

Disclosure of Usage of Generative Artificial Intelligence	v
List of Figures	xii
List of Tables	xv
Acronyms	xvii
1 Introduction	1
1.1 Context and Motivation	1
1.2 Problem Statement and Goals	1
1.3 Solution Overview	2
1.4 Methodology	3
1.5 Contributions	4
1.6 Document Structure	4
2 Background	6
2.1 Model-Driven Engineering and Low-Code Platforms	6
2.2 DEMO and DISME	7
2.2.1 DISME Architecture	9
2.2.2 The System Modeler (Frontend)	10
2.2.3 The System Executor	12
2.3 Large Language Models (LLMs) and Generative AI	14
2.4 Domain-Specific Languages and Validation	14
2.5 AI Assistance in Commercial Low-Code Platforms	15
3 State of the Art	16
3.1 Systematic Review	16
3.2 LLM Integration in Low-Code Platforms	17
3.2.1 Direct Generation and Copilots	17
3.2.2 Retrieval-Augmented Generation (RAG)	17

3.2.3	Agent-Based Generation and Repair	18
3.3	Intermediate Representations	18
3.3.1	Language-Based Representations	18
3.3.2	Structured Data Representations	19
3.4	Reliability and Validation	19
3.4.1	Semantic Hallucination	19
3.4.2	Automated Repair	20
3.4.3	Human-in-the-Loop Validation	20
3.5	Evaluation of AI-Assisted Development	20
3.5.1	Evaluation Methods	21
3.5.2	Efficiency and Reliability	21
4	System Proposal	22
4.1	Prior Work on DISME and Limitations	22
4.1.1	Modelling in DISME	22
4.1.2	The Modelling Barrier	24
4.1.3	Design Implications	24
4.2	System Requirements	25
4.2.1	Stakeholders	25
4.2.2	Functional Requirements	25
4.2.3	Non-functional Requirements and Scope	26
4.3	Proposed Solution Architecture	27
4.3.1	Architectural Overview	27
4.3.2	Phased Generation and Context Construction	28
4.3.3	Deterministic Validation and User Review	29
4.3.4	Integration with DISME	30
5	Implementation	31
5.1	System Overview	31
5.1.1	Architecture and Components	31
5.1.2	Models and Execution Environment	32
5.1.3	Workflow	32
5.2	Business Description (Phase 0)	33
5.2.1	Interview	34
5.2.2	Draft and Review	34
5.3	State and Process Models (Phases 1 and 2)	35
5.3.1	State Model	35
5.3.2	Process Model	36
5.3.3	Model Validation	39
5.4	Queries (Phases 3 and 3b)	40
5.4.1	Form Queries	40

5.4.2	Action Rule Queries	41
5.4.3	Query Paths and Validation	42
5.5	Action Rules (Phase 4)	43
5.5.1	Rule Interview and Generation	43
5.5.2	Rule Representation and Blocks Catalogue	45
5.5.3	Schema-Constrained Generation	47
5.5.4	Rule Validation	49
5.5.5	Blockly Bridge	50
5.5.6	Rule Example	51
5.6	DISME Integration	52
5.6.1	Model Import	53
5.6.2	Query Import	54
5.6.3	Blockly XML and Action Rule Drafts	55
5.6.4	Form Creation	55
5.6.5	Execution in DISME	58
5.7	System Creation Example	59
5.8	Evolution from the Initial Proposal	61
6	Validation	64
6.1	Context	64
6.2	Study Planning	65
6.2.1	Goals	65
6.2.2	Participants	66
6.2.3	Study Design	67
6.2.4	Measures	69
6.3	Execution	75
6.3.1	Preparation and Pilot	75
6.3.2	Procedure	76
6.4	Analysis	80
6.4.1	Data Preparation	80
6.4.2	Analysis Procedure	80
6.4.3	Results	82
6.5	Discussion	90
6.5.1	Interpretation of Results	90
6.5.2	Threats to Validity	93
6.6	Summary	94
7	Conclusions	95
7.1	Conclusions and Contributions	95
7.2	Future Work	96
	Bibliography	98

Appendices

A SLR	108
Large Language Models in Low-Code and Model-Driven Platforms: A Systematic Literature Review	108
A.1 Introduction	109
A.2 Background and Related Work	110
A.2.1 Low-Code and Model-Driven Engineering	110
A.2.2 Large Language Models in Software Engineering	111
A.2.3 The Intersection: AI-Augmented Modelling	111
A.2.4 Related Work	112
A.3 Methodology	113
A.3.1 Goals and Research Questions	114
A.3.2 Search Strategy and Study Selection	115
A.3.3 Data Extraction and Synthesis	119
A.4 Results and Analysis	119
A.4.1 Study Metadata and Research Trends	120
A.4.2 RQ1: How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?	124
A.4.3 RQ2: What types of intermediate or structured representations are used?	127
A.4.4 RQ3: What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?	128
A.4.5 RQ4: How are these approaches evaluated, and what benefits or limitations are reported?	129
A.5 Discussion	131
A.5.1 Common Architectures for Integrating AI	131
A.5.2 The Importance of Intermediate Formats	132
A.5.3 Ensuring Reliability in AI-Generated Models	133
A.5.4 Balancing Speed with the Need for Review	134
A.5.5 Current Trends and Industry Interest	134
A.6 Threats to validity	135
A.6.1 Construct validity	135
A.6.2 Internal validity	135
A.6.3 External validity	135
A.6.4 Conclusion validity	136
A.7 Conclusions and future work	136
B Validation Materials and Session Evidence	138
B.1 Artefact Quality Rubric	138
B.2 Session-Level Results	139

C	Study Instruments	141
C.1	NASA-TLX Questionnaire	141
C.2	Final Questionnaire	142
C.2.1	Previous Experience	142
C.2.2	System Usability Scale	142
C.2.3	Study-Specific Statements	143
C.2.4	Open Questions	143
C.2.5	Remote-Access Condition	144
D	System Walkthrough	145
D.1	Start with the business description	145
D.2	Inspect the proposal and request a correction	146
D.3	Accept the correction and confirm the scope	147
D.4	Define and review the State Model	149
D.5	Confirm roles and the Process Model	150
D.6	Import the structural models	151
D.7	Review form-choice and logic queries	152
D.8	Specify and inspect the Submit Review rule	154
D.9	Correct form fields or request a behaviour change	155
D.10	Approve the reviewed rules	156
D.11	Import the draft and activate it in DISME	157
D.12	Check the rule blocks before saving	158
D.13	Prepare the Form and the reviewer application	160
D.14	Submit a review and verify the stored result	161
D.15	Inspect the records created and updated	162
 Annexes		
I	DISME's EBNF Grammar	164

LIST OF FIGURES

1.1	Wizard overview and DISME integration	3
2.1	The ontological building blocks of DEMO	8
2.2	The complete ontological model of an organisation	8
2.3	DISME functional environments: System Modeler (Design-Time) and System Executor (Run-Time). From [8].	10
2.4	DISME technology stack and integration points. From [8].	11
2.5	The Forms Manager interface using Form.io.	12
2.6	DISME Dashboard showing processes, pending tasks, task statistics, and dele- gations.	13
4.1	The Action Rules Manager interface with the integrated Blockly editor. From [8].	23
4.2	Action Rule specification example using Blockly. From [8].	23
4.3	Proposed solution architecture overview	28
5.1	Review of a generated State Model in the wizard.	37
5.2	Review of the generated rule for <i>Submit Review</i> in the wizard.	52
5.3	Imported draft of <i>Submit Review</i> : form fields and score validation in the DISME Action Rules Manager.	56
5.4	Continuation of the imported <i>Submit Review</i> draft: automatic assignments, status changes, and confirmation message.	57
5.5	Execution Form for <i>Submit Review</i> in DISME.	62
6.1	Task completion status for the seven assessed tasks.	83
A.1	Distribution of the 69 initial search results across the selected digital libraries.	116
A.2	Study selection flow from database searches and snowballing to the final sample.	117
A.3	Distribution of selected studies per year (2022–2025).	121
A.4	Distribution of selected studies by publication venue type.	122

A.5	Evolution of publication venues over the years.	122
A.6	Distribution of studies by author affiliation type.	123
A.7	Geographical distribution of selected studies on a world map.	123
A.8	Number of authors per country, based on their institutions.	124
D.1	Opening exchange from the preserved conversation: the business description and the next question.	146
D.2	Initial entity proposal, including Decision as a separate entity.	146
D.3	Correction request specifying the removal and the three additions, before preparing the revised proposal.	147
D.4	Revised eight-entity proposal with the controls to accept it or retain the previous version.	148
D.5	After accepting the correction, the overview can be confirmed and the wizard can continue.	148
D.6	Final State Model review. Review contains four ordinary properties and two references to existing entities.	149
D.7	Role review with the three responsibilities and the correction and confirmation controls.	150
D.8	Final Process Model review with Review Process expanded: Submit Review, its roles, result and start/end flags.	151
D.9	Structural import screen, with the API connection and import control.	152
D.10	Saved import completion state and the control to continue to Phase 3.	152
D.11	SubmittedArticles restricts the Article choices available when creating an Assignment.	153
D.12	Logic query with Article and Reviewer as runtime parameters and Status fixed to Pending.	153
D.13	Submit Review review screen: form fields, references from Assignment, status changes and notification.	154
D.14	Manual field editor: Score and Recommendation are required; Comments is optional.	155
D.15	Pending review: approval and problem-reporting controls, with import unavailable.	156
D.16	Approved Submit Review rule: the import action is now available.	157
D.17	Wizard screen for importing the reviewed rule as a draft.	158
D.18	Action Rules Manager controls. Load Draft opens the imported draft; Save Action Rule saves the active rule.	158
D.19	Form block with the required fields and the two score limits.	159
D.20	Automatic assignments bind the Review to the selected Assignment and set the two records to different statuses.	159
D.21	Form preparation after the drafts have been imported and saved as active rules in DISME.	160

D.22 Saved result: the required Form already existed, so no new Form was created.	160
D.23 The actual Submit Review Form immediately before submission, with Score 4, Recommendation Accept and the comment entered.	161
D.24 The submitted Review stores Score 4, Recommendation Accept, the entered comment and Status Submitted. Reviewer 2 and Article 1 match the selected Assignment.	162
D.25 The selected Assignment now has Status Completed and retains the same Reviewer and Article references. Its assignment date and deadline are also shown.	163

LIST OF TABLES

3.1	Research questions of the systematic literature review.	17
5.1	Model settings for the main interview and extraction calls.	33
5.2	Form components generated for each property type.	58
5.3	Creation and execution of the CONFERENCE example.	60
5.4	Changes made during implementation.	62
6.1	Guiding questions and main sources of evidence.	66
6.2	Materials used in the participant study.	68
6.3	Planned distribution of a participant session.	68
6.4	Responsibilities during the common workflow.	69
6.5	Measures collected during the study.	70
6.6	Planned criteria for interpreting the study results.	74
6.7	Participant tasks and conditions for completion.	78
6.8	Previous familiarity and experience of the 19 analysed participants.	82
6.9	Assistance recorded for each participant task.	84
6.10	Duration of the common workflow in decimal minutes.	84
6.11	Main interactions recorded during the common workflow.	85
6.12	Quality of the initial and confirmed artefacts.	85
6.13	Artefact quality by wizard phase.	86
6.14	Functional outcomes obtained in DISME.	86
6.15	Critical errors in sessions that passed the execution checks.	87
6.16	NASA-TLX ratings for the common workflow.	87
6.17	Responses to the study-specific statements.	88
6.18	Difficulties and needs identified in the qualitative records.	88
6.19	Improvements suggested by participants.	89
6.20	Topics followed during free exploration.	89
6.21	Results of the predefined interpretation criteria.	90
A.1	PICOC Analysis.	114

A.2	List of selected studies.	117
A.3	Data extraction form fields.	120
A.4	Integration Techniques of LLMs in Low-Code Platforms	125
A.5	Interaction Paradigms between User and LLM	126
A.6	Types of Artefacts Generated by the Integration	126
A.7	Formal and Language-Based Representations (Language-Based)	128
A.8	Structural and Data Representations (Data-Structure Based)	128
A.9	Correctness Assurance Mechanisms	129
A.10	Evaluation Methodologies	130
A.11	Reported Benefits and Limitations	131
B.1	Artefact quality rubric for the reduced conference scenario.	138
B.2	Task, quality, assistance, import, and execution results for the 19 analysed sessions.	140
C.1	NASA-TLX items presented through the study HTML.	141
C.2	Participant profile questions.	142
C.3	System Usability Scale items used in the final questionnaire.	143
C.4	Study-specific statements used in the final questionnaire.	143
C.5	Open questions used in the final questionnaire.	143
I.1	EBNF Grammar Definition from [8]	164

ACRONYMS

ACM	Association for Computing Machinery
AI	Artificial Intelligence
AM	Action Model
AOM	Adaptive Object Model
API	Application Programming Interface
BNF	Backus-Naur Form
CM	Construction Model
DEMO	Design and Engineering Methodology for Organizations
DISME	Dynamic Information System Modeller and Executor
DSL	Domain-Specific Language
DSR	Design Science Research
EBNF	Extended Backus-Naur Form
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IR	Intermediate Representation
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LCD	Low-Code Development
LCDP	Low-Code Development Platform

LLM	Large Language Model
M2M	Model-to-Model
M2T	Model-to-Text
MDE	Model-Driven Engineering
ML	Machine Learning
MVC	Model-View-Controller
NASA-TLX	NASA Task Load Index
NLP	Natural Language Processing
PHP	PHP: Hypertext Preprocessor
PICOC	Population, Intervention, Comparison, Outcome, Context
PM	Process Model
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
RQ	Research Question
SLR	Systematic Literature Review
SM	State Model
SPA	Single-Page Application
SSH	Secure Shell
SUS	System Usability Scale
SysML	Systems Modeling Language
UML	Unified Modeling Language
WYSIWYG	What You See Is What You Get
XMI	XML Metadata Interchange
XML	Extensible Markup Language

INTRODUCTION

1.1 Context and Motivation

Through visual modelling and configuration, Low-Code Development Platforms (LCDPs) support software development without requiring extensive manual coding [5]. Dynamic Information System Modeller and Executor (DISME) is a low-code platform for modelling and executing organisational information systems [6]. It combines graphical diagram editors and visual rule programming with the Design and Engineering Methodology for Organizations (DEMO) enterprise engineering methodology [7]. Its Action Rules Manager uses Blockly, a visual block-based editor, to define system behaviour [8].

However, modelling an organisational system in DISME still requires users to select and configure the necessary components through the System Specifier and the visual editors. Users must understand how business requirements are represented through entities, properties, roles, processes, transactions, and Action Rules. These demands can create a steep learning curve for users who know the business domain but are unfamiliar with the platform.

In this dissertation, we propose to use Large Language Models (LLMs) to assist this modelling process. Users describe the intended system in natural language, and an LLM-assisted interface supports the generation of an initial DISME specification. Users can then review the proposed models and rules and request corrections before importing them into the platform. This approach is intended to help users express their business requirements without having to construct each model element manually through DISME's modelling interfaces.

The main objective is to help users create an initial DISME specification from a natural-language description of their business requirements, using an LLM-assisted approach.

1.2 Problem Statement and Goals

Creating a system in DISME requires users to translate business requirements into models and rules. They must define entity types, properties, roles, processes, and transactions,

and express the required behaviour through Action Rules. This work involves selecting and configuring model elements through the diagram editors, System Specifier, and Action Rules Manager [8]. The problem addressed in this work is the modelling effort this process requires: users must understand how to represent the intended system using the platform's concepts and manually configure its models and rules.

The aim of this work is to support users in creating system specifications for DISME. We propose an LLM-assisted interface through which users describe the intended system in natural language and review and correct the generated specification. The use of LLMs to support modelling has also been studied in low-code platforms [9].

The use of LLMs also introduces the risk of incorrect models [10]. In DISME, errors may include invalid references, unsupported operations, or missing requirements. Since the System Executor uses the models stored in the database at run time, these errors can affect its behaviour. The proposed system must therefore check the generated models and rules against DISME's constraints and allow users to review whether they reflect the intended business requirements.

The evaluation examines whether users can create and review an initial specification and whether the confirmed models and rules can be imported and used in DISME. It assesses their correctness, completeness, and behaviour during execution. An empirical study with users also records task completion, time, workload, usability, required assistance, and observed difficulties.

1.3 Solution Overview

The proposed solution is a wizard that helps users describe a business system and build an initial specification for DISME. It collects information through interviews in natural language and uses LLMs to generate the State Model, Process Model, queries, and Action Rules. The wizard also supports the creation of forms.

The State Model defines the entity types, their properties, and the relationships between them. The Process Model describes the processes, their transactions, and the roles responsible for them. Queries select and retrieve records needed by forms and Action Rules. Action Rules define the conditions and actions that control transaction execution. Forms collect information entered by users during execution. Figure 1.1 shows the main steps and the interaction with DISME.

The wizard builds the specification in phases, using the information confirmed in earlier phases as context for later ones. The generated models and rules use structured formats so that the wizard can check their types, fields, and references. Users review the results and can request corrections before confirming them. For Action Rules, the wizard also presents explanations in natural language to help users check whether the proposed behaviour matches their requirements.

Import takes place at different points in the workflow. The State and Process Models are imported before the queries and Action Rules that depend on them. Queries are imported

after review. Action Rules are converted into Blockly Extensible Markup Language (XML) and imported as drafts, which users inspect and save in the Action Rules Manager to activate them. The wizard can then create the forms needed by the saved rules. DISME uses the imported specification to run the system.

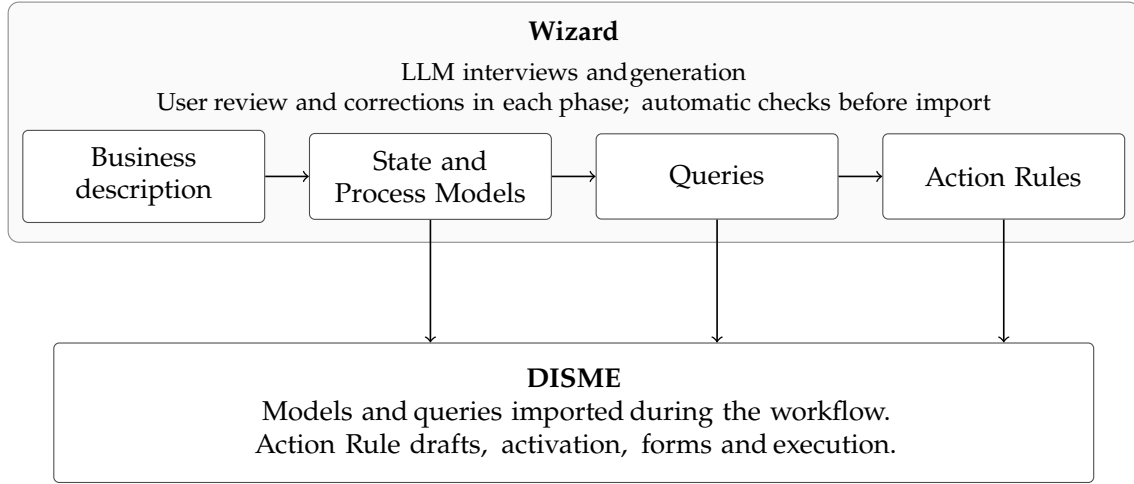


Figure 1.1: Overview of the wizard stages and their integration with DISME. Imports take place during the workflow, after the relevant checks and user confirmation.

1.4 Methodology

This work follows the Design Science Research (DSR) process proposed by Peffers et al. [11]. The process comprises six activities: problem identification and motivation, definition of the objectives, design and development, demonstration, evaluation, and communication.

The *Problem Identification and Motivation* activity is addressed in this chapter and further discussed in Section 2.2.1 of Chapter 2. Chapter 4, particularly Section 4.1.2, examines the knowledge of the platform and the algorithmic thinking required to define a system manually in DISME.

The *Definition of the Objectives* was guided by a Systematic Literature Review conducted as part of this work. The review followed the guidelines of Kitchenham and Charters [12]. Its findings helped define the requirements and design of the solution, including the use of intermediate representations, automatic checks, and user review. The review is summarised in Chapter 3 and presented in full in Appendix A.

The *Design and Development* activity is described in Chapters 4 and 5. Chapter 4 presents the requirements and the final conceptual design, while Chapter 5 explains how this solution was implemented and the choices made during development.

Development followed iterative cycles in which generated artefacts were compared with reference models and checked by deterministic validators. Identified errors led to

changes in prompts, extraction logic, validation, import, or runtime integration, followed by regression testing.

The solution is demonstrated through the system creation example presented in Chapter 5. Chapter 6 presents the evaluation method and results.

1.5 Contributions

DISME already provided the metamodel, modelling interfaces, Blockly block definitions and code generators, and the execution engine. This dissertation builds on these components and makes the following contributions:

- A Systematic Literature Review of how Large Language Models are applied in low-code development and model-driven platforms;
- A pipeline integrated with DISME that converts natural-language descriptions into initial specifications of entity types, properties, roles, processes, transactions, forms, queries, and Action Rules. The pipeline includes:
 - An intermediate representation for Action Rules, defined by a schema and checked against the confirmed models and supported DISME operations, with a loop for correcting errors;
 - A Blockly Bridge that converts this representation into Blockly XML, and an importer that sends the rules to DISME through its existing draft mechanism;
 - A wizard application that presents the generated models and natural-language explanations of Action Rules for review and correction before import. It runs separately from DISME’s System Specifier and Action Rules Manager.
- An empirical evaluation of the wizard and the generated models and rules, based on 19 analysed participant sessions using a reduced conference management scenario.

1.6 Document Structure

This dissertation consists of seven chapters.

Chapter 1 introduces the context, motivation, problem, and objectives of the research. It also presents an overview of the proposed solution, the methodology, and the contributions.

Chapter 2 introduces Model-Driven Engineering, low-code development, Domain-Specific Languages, and Large Language Models. It also describes the DEMO methodology, the DISME architecture, and the use of Artificial Intelligence (AI) in commercial low-code platforms.

Chapter 3 summarises the Systematic Literature Review on LLMs in model-driven and low-code environments. It presents the review’s research questions, study selection, data extraction process, and findings. The full review is included in Appendix A.

Chapter 4 presents the requirements and conceptual architecture of the proposed solution. It describes the phased generation process, context construction, deterministic validation, human review, and the different import paths used for structural artefacts and Action Rules.

Chapter 5 describes the implemented system, from the initial business description to the extraction and review of the State and Process Models, the definition of forms and queries, and the creation of Action Rules through a structured intermediate representation. It also covers automated validation and refinement, conversion to Blockly XML, import as drafts, and the evolution from the initial proposal.

Chapter 6 presents an empirical study with users to evaluate the usability of the wizard in a reduced conference management scenario. It also examines the assistance required by users and the quality, import, and execution of the generated models and rules in DISME.

Chapter 7 summarises the results, limitations, and possible future work.

BACKGROUND

This chapter introduces the concepts used in the proposed solution. It covers Model-Driven Engineering (MDE) and low-code platforms, the DEMO methodology, and the DISME architecture. It also discusses the use of LLMs to generate software artefacts, the limits of probabilistic output, and the roles of validation and human review. The final section compares this work with AI-assisted features in commercial low-code platforms.

2.1 Model-Driven Engineering and Low-Code Platforms

Model-Driven Engineering (MDE) puts models at the centre of the software development process. A model is an abstract representation of a system. By raising the level of abstraction, MDE allows developers to focus on business logic and domain concepts instead of implementation details [13]. Automated transformations can convert models into executable code or other models [14], reducing the manual work between design and implementation.

MDE and Low-Code Development Platforms both rely on abstractions and visual models [15]. Low-code platforms provide visual environments in which users build applications through configuration and reusable components, reducing the need for conventional programming [5]. This can reduce development effort and allow *citizen developers*, users without a professional software development background, to participate in application development [16, 17].

Reducing manual coding does not remove the need to understand a platform’s concepts and constraints. Luo et al. [18] found different views among practitioners about the benefits and limitations of low-code development. Some considered it more approachable than conventional programming, while others reported a learning curve and the need for custom code when requirements became complex. Users without a professional software development background may also encounter difficulties with requirements, design, testing, and other software engineering activities [19]. Domain knowledge must still be translated into formal specifications, including entities, relationships, process flows, and business rules. This difference between natural-language descriptions and formal

system specifications motivates the study of natural-language interfaces for modelling [20].

2.2 DEMO and DISME

The DISME platform is based on the Design and Engineering Methodology for Organisations (DEMO). This section summarises the principles needed to understand how the platform represents and executes organisational systems.

Based on the Ψ -theory (Performance in Social Interaction) [21], DEMO focuses on the acts performed by an organisation's actors rather than on data flows or functional decomposition. It distinguishes between Production Acts (**P-acts**), which bring about Production Facts (**P-facts**), and Coordination Acts (**C-acts**), through which actors make and honour commitments about those results. For example, delivering an order is a P-act, while the order having been delivered is the resulting P-fact.

A *Transaction* organises these acts and facts. It has an initiator, who requests a result, and an executor, who produces it. During the *Order Phase*, the two parties agree on what is to be produced. In the *Execution Phase*, the executor carries out the P-act and brings about the P-fact. During the *Result Phase*, the executor states that the result has been produced and the initiator accepts it.

The basic transaction pattern includes four C-acts, *request*, *promise*, *state*, and *accept*, and one P-act, *execute*. In DISME, the transaction progresses through the states *Requested*, *Promised*, *Executed*, *Declared*, and *Accepted*. The state *Declared* corresponds to the *state* act. The Execution Engine manages these states, and an Action Rule can be associated with a state to define the system's behaviour at that point.

A DEMO ontological model has four aspect models. The Construction Model (CM) defines transaction types, actor roles, and the relationships between them. The Process Model (PM) covers conditional and causal relationships between transaction steps. The Action Model (AM) specifies the business rules that guide the actors, while the State Model (SM) defines object classes and facts. Figure 2.2 shows how these models are related.

Terminology varies across different versions of DEMO. The terminology shown in Figure 2.2, based on Dietz [21], uses Construction Model and State Model. Dietz and Mulder [7] instead use Cooperation Model and Fact Model. This dissertation uses Construction Model and State Model, but uses Fact Model when describing DISME's diagram editor.

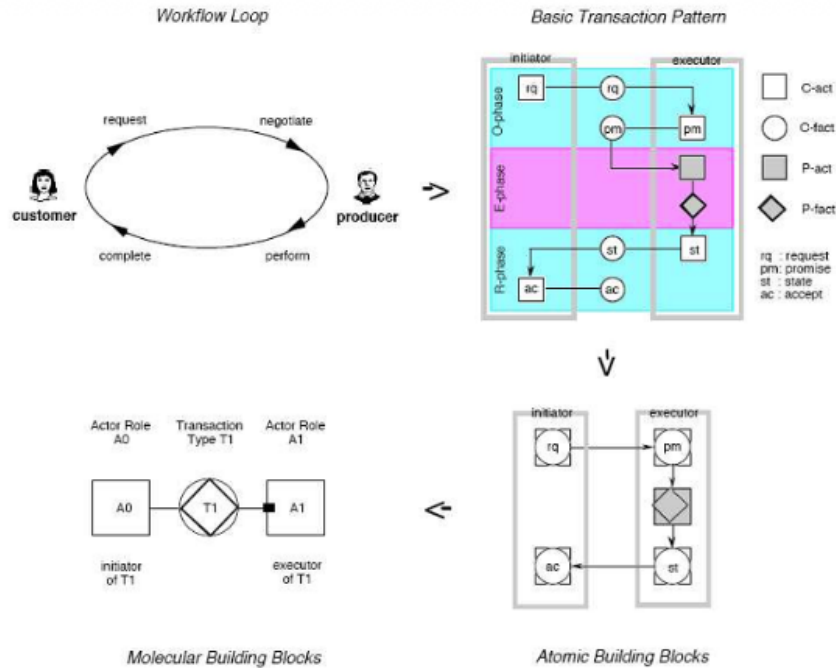


Figure 2.1: The ontological building blocks of DEMO, including the basic transaction pattern. From [22].

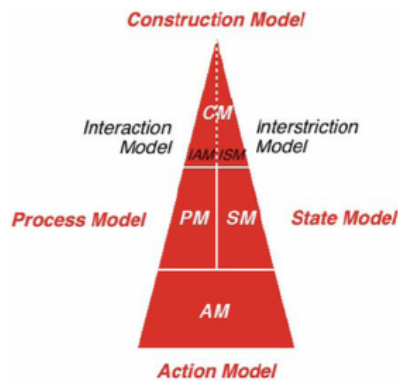


Figure 2.2: The complete ontological model of an organisation, composed of the four aspect models: Construction (CM), Process (PM), Action (AM), and State (SM). From [21].

DISME applies this approach in a low-code environment [6]. Rather than generating static code as its main output, it follows the Adaptive Object Model (AOM) architectural style [23]. The model's structure and behaviour are stored as data and interpreted at runtime.

The platform is divided into the System Modeler and the System Executor. In the System Modeler, organisational engineers specify the system through forms, tables, diagrams, and Action Rules. This includes defining actor roles, transaction types, entity

types, properties, and Causal Links between transaction steps [8]. A Causal Link specifies that one transaction step causes another, such as the request of a subsequent transaction. The System Executor then interprets the model to support the execution of organisational processes.

For example, in a conference system, the System Modeler can define an *Article* entity type, while individual articles of that type are created during execution as users submit them.

2.2.1 DISME Architecture

The DISME platform is the technological base for this research. DISME is a low-code, open-source platform for modelling and executing organisational systems based on the Design and Engineering Methodology for Organisations (DEMO) [6, 7]. DISME does not generate static code for applications. Instead, it views an organisation as a *living system*, allowing changes to processes to be immediately reflected in the runtime environment [6].

DISME is designed using a distributed Model-View-Controller (MVC) architecture within a *client-server architecture* [8]. This design separates the client-side interface from the server-side application logic and data access. Three architectural aspects of DISME must be examined to understand the platform's capabilities and the integration points for the proposed LLM solution: the Adaptive Object Model, the Type Square Pattern, and the underlying technology stack.

2.2.1.1 Design-Time and Run-Time Separation

The Type Square Pattern allows for a clear distinction between an organisation's design and its operational execution [6]. Figure 2.3 illustrates the two functional environments used in this approach:

- **System Modeler** (Design-Time): This is the design-time configuration environment where organisational engineers can define their system using specific types of components, such as the System Specifier, Diagram Manager, Action Rules Manager, and the Form Manager, which allow the engineer to specify the parameters of the organisation's structure [6].
- **System Executor** (Run-Time): This environment handles the daily management of an organisation's processes by interpreting the models created in the System Modeler. Specifically, the Execution Engine dynamically interprets the stored Action Rules to manage transaction states and enforce workflow logic without compilation.

2.2.1.2 Technology Stack and Integration Points

DISME uses a decoupled technology stack. Figure 2.4 shows the technologies relevant to its operation and to the wizard's integration:

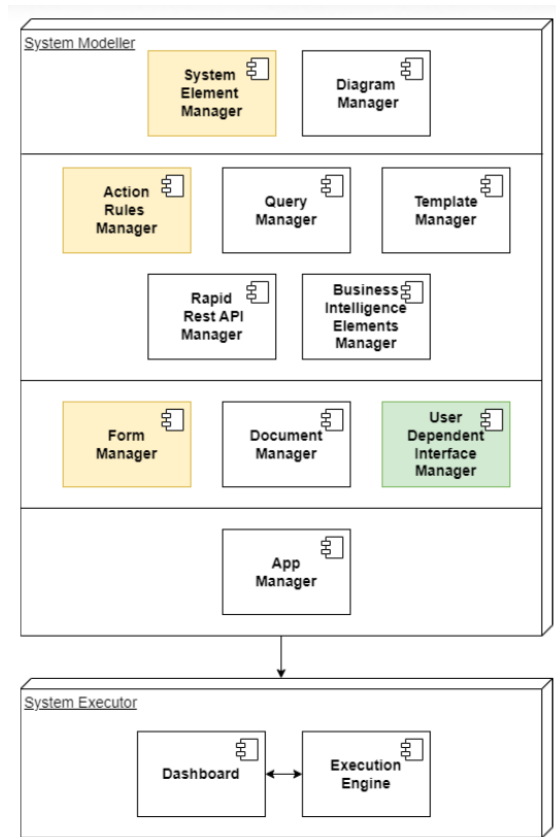


Figure 2.3: DISME functional environments: System Modeler (Design-Time) and System Executor (Run-Time). From [8].

- **Server (Backend):** The server-side implementation is done in PHP, utilising the Laravel framework [8]. Its implementation uses an MVC architecture pattern and exposes the system's capabilities via an Application Programming Interface (API) based on Representational State Transfer (REST). The API exposes model configuration and runtime operations and provides the main integration point used by the wizard.
- **Client (Frontend):** The client-side user interface is developed using TypeScript and the Angular framework [6]. The client is a Single-Page Application (SPA) that communicates asynchronously with the server through the API.
- **Database (Data Store):** Both the metadata related to the models and the operational execution data are stored within a MySQL relational database.

2.2.2 The System Modeler (Frontend)

The System Modeler is the configuration interface for organisational engineers to define the behaviour and structure of the system. Technically, it is the client-side interface from which the user populates the Adaptive Object Model stored in the database. The Modeler

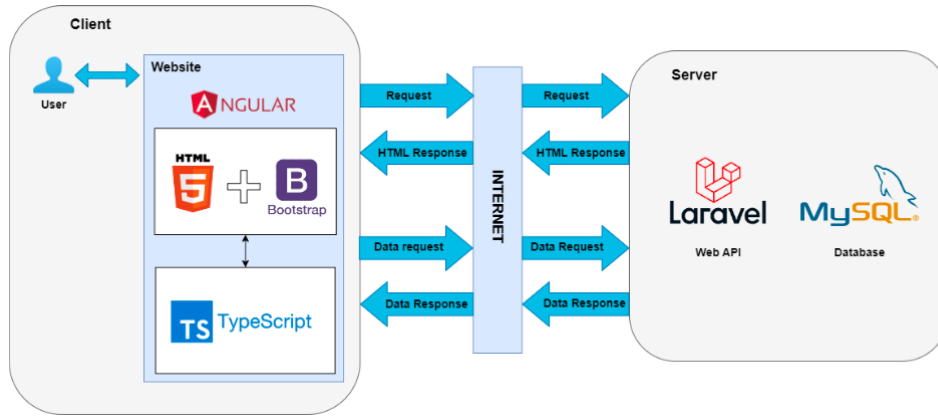


Figure 2.4: DISME technology stack and integration points. From [8].

separates the specification process into components for structural design, business logic, and user interfaces.

2.2.2.1 Diagram and Structure Management

The initial step in defining an organisation within DISME usually requires the development of a visual representation of its processes. A specialised version of draw.io (an open-source diagramming tool) [24] called the Diagram Manager is integrated into the DISME platform, which enables users to develop their Process Model and Fact Model visually, using DEMO-specific notation [6]. Although the tool can parse these visualisations to produce the corresponding database table rows, the tool relies on users to develop syntactically correct models based upon the methodologies themselves.

Complementing the visual diagrams, the System Specifier contains a table view of the model details. Within this environment, users specify entity types, properties, and transaction types explicitly. Users also configure the Causal Links and Waiting Links among the various transaction steps through the process of wiring the system. This process requires a detailed understanding of the platform's concepts to configure the appropriate dependency relationships between transaction steps.

2.2.2.2 Forms Management

The user interface that is presented to end-users during runtime is defined through the Forms Manager. It also integrates the Form.io framework to make it easy to use a What You See Is What You Get (WYSIWYG) editor [25]. Within this editor, users can drag and drop various input elements and layout elements onto a canvas to create their own form templates and enter their own labels and validation criteria.

As shown in Figure 2.5, forms are associated with Actions and their Action Properties. When the Execution Engine reaches an action that requires user input, it displays the associated form [6].

Available Actions						
Create Form						
Forms Management						
ID	Form Name	Transaction Name	Action Name	Updated at	Created at	Available Actions
1	Create Branch Form	Creation of Branch	Create Branch Action	2022-07-28 12:25:58	2022-07-28 12:25:58	Edit Delete Render
2	Create Car Type Form	Creation of Car type	Create Car Type Action	2022-07-28 12:26:45	2022-07-28 12:26:45	Edit Delete Render
4	RAC Request Form	Rental Completing	RAC Request Action	2023-05-03 18:17:14	2022-07-28 14:10:46	Edit Delete Render
5	Rental Payment Price Insertion Form	Invoice Paying	Rental Payment Price Insertion	2022-07-28 14:13:07	2022-07-28 14:13:07	Edit Delete Render
6	Rental Payment Execution Form	Invoice Paying	Rental Payment Execution	2023-03-15 09:51:05	2022-07-28 14:15:08	Edit Delete Render
8	Car Drop Off Execution Form	Car Returning	Car Drop-Off Execution	2022-07-28 19:11:06	2022-07-28 19:11:06	Edit Delete Render
9	Create Car Form With Features	Creation of Car	Create Car Action	2023-05-03 18:17:14	2022-11-23 09:39:42	Edit Delete Render
18	Admin Specification	Admin Details Specification	Admin Details Specification	2023-05-03 18:17:14	2023-03-15 15:52:05	Edit Delete Render
19	HP Specification	Health Professional Details Specification	Health Professional Details Specification	2023-03-15 15:52:43	2023-03-15 15:52:43	Edit Delete Render
21	Admin Edition	Admin Details Edition	Admin Edition	2023-03-15 16:39:57	2023-03-15 16:39:57	Edit Delete Render

Figure 2.5: The Forms Manager interface using Form.io.

2.2.3 The System Executor

The System Executor is the runtime operating environment of the DISME platform. While the System Modeler provides the structure of the organisation, the System Executor interprets that definition to execute the information system. It interprets the models stored in the database at runtime without generating or compiling application code [6]. In a technical sense, the System Executor is mainly located at the server side (backend), utilising the *Laravel* framework. The server exposes a RESTful API that serves as a bridge between the database (that stores the Adaptive Object Model) and the client-side user interface.

2.2.3.1 The Execution Engine

The main part of the System Executor is the Execution Engine. The Execution Engine controls how each organisational process is executed according to its transaction states and Action Rules [8].

The Execution Engine represents each transaction using the five DEMO states stored in DISME: *Requested*, *Promised*, *Executed*, *Declared*, and *Accepted*.

As described by [8], the Execution Engine performs four key functions during execution:

- **State Management:** It keeps track of the status of each Transaction Instance (for example, if a Rental Request is "Requested" or "Promised").

- **Rule Interpretation:** Just before the state change is stored in the database, the Execution Engine intercepts the request and validates it against the Action Rules defined in the Modeler.
- **Causal Linking:** Based upon the completion of prior tasks, it triggers downstream transactions automatically as defined within the Causal Links configuration of the System Specifier.
- **Logging and Traceability:** It records execution information and transaction state changes.

2.2.3.2 Runtime Interface (Dashboard)

The server provides the data used by the User Dashboard on the frontend. The User Dashboard is a context-sensitive graphical user interface that presents users with their current task status and enables them to start new processes or perform actions associated with pending ones.

In addition, it functions as an interface layer to the Execution Engine. When an Action Rule reaches an action that requires user input, such as `user_input` or `edit_entity_instance`, the dashboard presents the associated form. After the user submits the form, the dashboard sends the entered data to the server through the API, allowing the execution to continue.

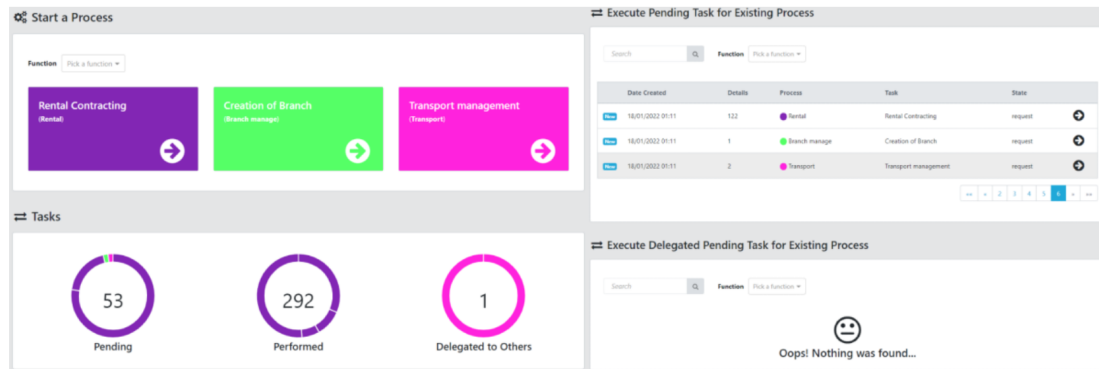


Figure 2.6: DISME Dashboard showing processes, pending tasks, task statistics, and delegations.

Defining a system in DISME requires knowledge of the organisational domain and the concepts used by the platform. Entity types and properties represent part of the State Model. Actor roles, transaction types, and process entry and exit information represent parts of the Construction and Process Models. Action Rules represent part of the Action Model, while forms and queries support user interaction and runtime execution.

This dissertation addresses the gap between a user's description of an organisation and the formal specification required by DISME. This is a platform-specific instance of

the abstraction gap between natural-language descriptions and formal representations discussed in [20]. Chapter 4 presents the solution designed to address this gap.

2.3 Large Language Models (LLMs) and Generative AI

Large Language Models are generative AI models trained on large collections of text. Models for software engineering may also be trained on source code [26, 27]. Many LLMs use the Transformer architecture introduced by Vaswani et al. [28]. An autoregressive LLM uses the preceding context to estimate the probability of the next token. This supports tasks such as translation and summarisation. This dissertation uses LLMs to convert natural language into system specifications and other structured software artefacts.

LLMs have been used for documentation, testing, design, and requirements engineering [26]. They can also turn natural-language descriptions into source code, although performance varies with the model, task, and evaluation setting [27]. In this work, an LLM interprets a business process description and proposes parts of the structural and behavioural specification. The output is an initial specification for review, not a finished system.

The prompt influences the generated output. A system prompt sets the instructions, while user messages provide the task content. Few-shot prompting adds examples of expected inputs and outputs [29]. Temperature influences variation during generation. Lower values tend to produce less variable results, but do not guarantee correctness [30].

Structured generation limits the output format. For JavaScript Object Notation (JSON), JSON Schema can define required fields, data types, allowed values, and nested structures [31]. When supported, an inference engine can use the schema during token generation to keep the output within that structure [32]. This can reduce syntax errors, but does not ensure that references are valid or that the result matches the user’s intent. The output still needs deterministic validation and human review.

Even with these controls, generated artefacts may be structurally valid while remaining semantically incorrect or incomplete. Software engineering research shows that hallucinations can produce code that appears valid but is functionally incorrect [26]. In DISME, this may result in invalid relationships, references to actor roles or transactions that do not exist, or the omission of requirements provided by the user.

These errors matter because the System Executor interprets the specifications stored in DISME at runtime. Generated content must therefore be checked against the platform’s constraints, and its meaning must be reviewed by the user before import.

2.4 Domain-Specific Languages and Validation

Domain-Specific Languages (DSLs) can reduce the distance between human intent and formal specifications. Fowler [33] defines a DSL as a programming language of limited expressiveness focused on a particular domain. General-purpose languages such as Java

or C# address a broad range of computational problems, whereas DSLs provide concepts and notation for a specific domain. In model-driven engineering, a DSL allows domain experts to work with concepts such as Transactions and Actor Roles instead of low-level implementation details. A formal grammar and supporting tools can prevent some syntactically invalid constructions, but cannot guarantee that a specification represents the intended system.

DISME uses domain concepts derived from DEMO and a formal language for Action Rules defined through an Extended Backus-Naur Form (EBNF) grammar [8]. These concepts and rules determine what the platform can represent and how it interprets each element. Because DISME follows an AOM architecture, it stores the system's structure and behaviour as data and interprets them at runtime [6, 23]. Changes to an active model do not require recompilation and can affect later execution. An incorrect specification may therefore create an invalid configuration or unintended process behaviour.

A staging mechanism can keep generated content inactive until it has been reviewed. Its position in a workflow depends on how the target platform stores and activates each type of artefact. Structural and behavioural artefacts may therefore require different review and activation points. Chapter 4 explains how this principle is applied in the proposed solution.

Automated checks can assess defined platform constraints, while human review is needed to determine whether the specification represents the intended business meaning.

2.5 AI Assistance in Commercial Low-Code Platforms

Commercial low-code platforms provide related features. According to its documentation, OutSystems Mentor can generate an application, including its data model, business logic, and user interface, from a requirements document or natural-language prompt [34]. Mendix Maia uses natural-language input to create or modify domain models, pages, microflows, and workflows [35]. Copilot for Power Apps can generate an application and its Dataverse model, including tables and relationships, from a natural-language description [36].

Each platform provides a way to review and modify generated content. OutSystems allows users to inspect and refine the application and its logic [34]. Mendix presents Maia's proposed changes for review before they are applied [35]. Power Apps allows users to alter tables and relationships before their creation [36]. These review mechanisms were not designed for the specific constraints of DEMO and DISME.

Pattnayak and Bohra report limitations in the flexibility and reliability of LLM-based zero-code tools [37]. Review and correction mechanisms differ between platforms, but they do not remove the risk of errors in generated applications.

This work applies natural-language generation to the DISME metamodel and execution environment. Unlike assistants for general-purpose low-code platforms, the proposed solution must account for the concepts and constraints of DEMO and DISME. Its design is presented in Chapter 4.

STATE OF THE ART

3.1 Systematic Review

A Systematic Literature Review was conducted to identify how LLMs are integrated into low-code and model-driven systems, the intermediate representations used between natural-language input and generated artefacts, and the mechanisms used to validate those artefacts.

Appendix A presents the review protocol, search string, inclusion and exclusion criteria, study-selection process, data-extraction form, and complete results. This chapter synthesises the findings most relevant to the proposed solution.

The review followed the guidelines for Systematic Literature Reviews in Software Engineering proposed by Kitchenham and Charters [12]. The search was conducted on 5 December 2025 across four digital libraries (*ACM Digital Library*, *IEEE Xplore*, *ScienceDirect*, and *Springer Link*), focusing on the intersection of three key concepts: "Low-Code/Model-Driven," "Large Language Models," and "Generation/Automation".

The database search returned 69 records. After removing four duplicates, 65 records remained for screening. Assessment against the inclusion and exclusion criteria excluded 37 records, leaving 28 studies. Forward and backward snowballing added seven studies, giving a final set of 35 selected studies. The search covered publications from 2020 to 2025, and the selected studies were published between 2022 and 2025. The analysis of these studies was guided by four **Research Questions (RQs)** that structure the discussion in the following sections of this chapter, represented in Table 3.1.

The findings informed the architecture presented in Chapter 4, particularly its use of structured intermediate representations, deterministic validation, and human review.

Table 3.1: Research questions of the systematic literature review.

ID	Research Question
RQ1	How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?
RQ2	What types of intermediate or structured representations are used between natural language inputs and executable artefacts?
RQ3	What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?
RQ4	How are these approaches evaluated, and what benefits or limitations are reported in AI-assisted Low-Code development?

3.2 LLM Integration in Low-Code Platforms

The reviewed studies describe different approaches to integrating LLMs into low-code and MDE. The studies show an increase in architectural complexity for LLMs used as external assistants, as well as increasing degrees of contextualisation and integration with the system. These architectures have been identified at three levels based on their level of autonomy and context-awareness: direct generation (stateless), Retrieval-Augmented Generation (RAG), and agentic feedback loops.

3.2.1 Direct Generation and Copilots

The first and simplest approach found in the literature is *Stateless Prompt Engineering* which can be used as *Copilot* or *Side-by-Side Assistant architectures*. This model was shown to work in studies of Wang et al. [38] and Sun et al. [39]. In this one-shot process, the user provides a natural-language description and the system produces an artefact (code or model) on that same input in a single pass.

Although implementing these architectures is easy as no changes need to be made to the underlying platforms [40], many serious limitations exist according to the existing literature. According to Tinnes et al. [10], the lack of knowledge about the larger project state or the model context causes the LLMs to fail to correctly predict how the artefact evolves, which creates a problem that is referred to as hallucination. This happens when the generated artefacts can have the correct syntax, but can be wrong from a functional standpoint when compared to other parts of the overall application, requiring the user to perform manual verification.

3.2.2 Retrieval-Augmented Generation (RAG)

Many researchers have moved towards *Retrieval-Augmented Generation* to address limitations caused by stateless generation. In other words, RAG architectures are able to

inject dynamic context into the prompt before generation. This dynamic context may be anything from platform documentation, to database schema, to existing metamodels.

Tinnes et al. [10] further indicated that it is extremely important for industry-based systems to be able to provide an external source of context, so that the system understands the company's specific vocabulary and constraints when generating the code. In a similar manner, Xu et al. [41] developed a method called *Rewrite-Retrieve-Read*, where the model first retrieves the correct API definition prior to the generation stage to reduce the error rate related to specific domain constraints. Both of these approaches support the findings made by [42] and [43], which found that constraining the generation process with previously-retrieved context greatly reduces semantic errors in proprietary domain languages.

3.2.3 Agent-Based Generation and Repair

Some reviewed approaches use *Agentic Feedback Loop* and *Multi-Agent System*. These architectures divide generation, execution, verification, and correction into separate steps, using the result of one step as input to the next.

Liventsev et al. [44] evaluated a generate–execute–repair loop in which failed tests provide feedback for revising generated programs. Grishina et al. [45] extended this approach with a multi-agent process that synthesises candidates, executes them, formulates repair instructions, debugs them, and ranks them for further iteration. Both studies evaluate program synthesis.

To summarise, current trends indicate a transition toward *hybrid neuro-symbolic architectures*. The reviewed approaches combine probabilistic generation with deterministic validation. The trend is to use the LLMs as a creative hypothesis generator, while the deterministic engine of the Low-Code platform acts as a validator, separating the interpretation of intent from structural compliance [46, 47].

3.3 Intermediate Representations

One of the central conclusions of the systematic review is that the generation of complex models or binary files using LLMs can be difficult because of the structural hallucination phenomenon. This difficulty was identified in the literature as a significant barrier that could be addressed by using *Intermediate Representations (IRs)* as a connection point between the probabilistic nature of the AI and the rigid deterministic nature of engineering platforms. Through the analysis of the reviewed studies, the two main categories of IRs were established as follows: Language-Based Representation and Data-Structure Based Representation.

3.3.1 Language-Based Representations

The predominant category identified in the literature refers to the use of textual DSLs. Silva & Hildebrandt [48] proposed combining generative AI with symbolic models through a

formal, human-readable DSL to support reliable no-code development. The literature also identifies human readability as an important consideration when using LLMs, because LLM output is probabilistic and users need to inspect it before execution. To address this issue, Liu et al. [20] proposed the concept of "*Grounded Utterances*" where the LLM produces a structured natural language representation of its output, explaining what each operation does to reduce the cognitive barrier of understanding the output, while allowing non-technical users to detect logical errors. Similarly, Zhou et al. [47] showed that producing *pseudocode* or intermediate instructions before creating final visual blocks results in higher accuracy, as it isolates the business logic from the complexity of graphical rendering.

3.3.2 Structured Data Representations

While DSLs focus on readability, data-based representations allow for technical integration while preserving model topology. As indicated in studies conducted by Sun et al. [39] and Koh [42], JSON objects and structured commands can be used as intermediate representations because they are easy to parse and can be mapped from hierarchical structures to relational databases. The existing literature also identifies challenges with regards to generating diagrams, workflows and ensuring the consistency of the nodes and edges in the diagram. To ensure the preservation of this consistency, the studies of Chen et al. [49] and Tinnes et al. [10] suggest using serialised graph representations to help the AI maintain topological consistency and prevent the creation of "floating" edges or invalid cycles (and thus maintain the integrity of the metamodel).

3.4 Reliability and Validation

Although integration architectures and intermediate representations offer a practical approach to generating models, LLMs' probabilistic nature creates reliability concerns in software engineering [10]. The systematic review demonstrated that the probabilistic nature of LLMs is a major source of hallucinations, where generated code may be syntactically valid but functionally incorrect. The current state-of-the-art has identified three main methods to mitigate these risks of hallucination: constraint-based generation [50], automated verification loops [45] and human-in-the-loop validation [47].

3.4.1 Semantic Hallucination

The literature has highlighted a significant difference between syntactically valid and semantically correct code. Researchers, such as Netz et al. [50], have successfully used modern techniques (such as *Grammar Masking*) to generate code with nearly 100% syntactical validity through the restriction of token generation that violates the syntax of the language's grammar. However, this syntactical validity does not equate to usefulness.

Wang et al. [38] and Tinnes et al. [10] both reported that although LLM-generated code may compile correctly, it fails to accurately capture domain-specific business logic, or invents relationships that do not exist in the project (hallucinations). Finding and correcting these errors can take time, and this should be considered when assessing productivity gains.

3.4.2 Automated Repair

To eliminate the constant need for human intervention in order to reduce hallucination, the literature showed a shift from one-shot generation to *Self-Repair* cycles. This cycle is based on the idea that the system accepts the output from the LLM as non-final. Instead, the system sends the output of the LLM through a compiler or simulator, and if errors are detected, the system sends the error messages back to the LLM which then uses those to correct itself independently. Liventsev et al. [44] empirically found that the self-repair loop of *Synthesize-Execute-Debug* provides better results than the direct synthesis generation process and transforms the LLM into an autonomous agent, as opposed to simply providing a static answer.

3.4.3 Human-in-the-Loop Validation

While advances have been made in automated repair of systems, the reviewed studies continue to include human validation. For human validation to be effective, the system cannot simply provide large amounts of complex code or raw data in a format such as JSON files. This would induce excessive cognitive load, which is detrimental to the success of auditing. In an attempt to resolve this issue, staging mechanisms were identified: these involve separating all generated content within a draft area and providing readable descriptions and simple review interfaces before allowing the generated content to be integrated into the production system. This *Human-Centric AI* strategy ensures that the responsibility of the final decision regarding business logic remains in the hands of the human expert, while the AI performs the technical implementation. This is a design philosophy central to the proposed architecture for DISME.

3.5 Evaluation of AI-Assisted Development

The fourth research question asked how these AI-assisted approaches are evaluated in practice, and what benefits or limitations the literature reports. The reviewed studies split roughly into two groups on this point: one relies on quantitative measurement of productivity gains, the other runs qualitative studies with users and reports that generated output can still be semantically incorrect and needs validation effort before it can be trusted.

3.5.1 Evaluation Methods

Four evaluation methods appear across the reviewed studies: automated benchmarks against reference datasets, industrial case studies and pilots, controlled user studies, and focus groups or surveys that capture practitioner vision rather than measured results. Sun et al. [39] proposed a formula for efficiency, $E = (W_t \times \Delta Time) + (W_c \times \Delta Cost) + (W_e \times \Delta Errors)$, and applied it to production data from JD.com and China Unicom, reporting a 60% increase in development efficiency compared to the traditional workflow. Hagel et al. [9] and Malamas et al. [51] measured accessibility more directly, through metrics such as *Valid Model Rate* and time-on-task, and both found that AI-based tools lower the entry threshold for non-expert developers and reduce the time needed to produce a first-draft DSL specification.

A large-scale mapping study published after this review’s search cutoff, Zhang et al. [52], analysed 86 primary studies on LLMs applied to MDE published between 2022 and early 2026. They report that validation in this literature is centred on quantitative experimentation, yet 42% of the studies report no baseline comparison at all, and cost efficiency is measured in fewer than a quarter of them. These findings show that similar evaluation gaps also appear in a larger review.

3.5.2 Efficiency and Reliability

Productivity gains do not come free. Wang et al. [38] and Tinnes et al. [10] report semantic errors in generated models. The time needed to find and correct these errors should therefore be considered when assessing productivity gains, as discussed in Section 3.4. Some studies investigate automated repair to address these limitations [44, 45]. Liventsev et al. [44] found that repairing code and generating new alternatives together gave better results than using only one of these methods in their experiments. Their use of feedback to revise generated code is also relevant to this dissertation’s self-refinement loop (Section 5.5.4).

These findings motivate the closing argument of this chapter: without some form of intermediate validation between natural-language input and executable specification, gains in speed do not translate into gains in reliability. Chapters 4 and 5 address this directly. One item from the literature’s future-work agenda, the call for standardised, shared benchmarks for comparing tools, is only partially met by this dissertation’s own evaluator scripts, which serve as an internal measure of progress rather than a benchmark for the wider community.

SYSTEM PROPOSAL

This chapter presents the design of the proposed solution at a conceptual level. It defines the problem addressed by the wizard, its requirements, its main components, and the flow from a business description to artefacts that can be used by DISME. Chapter 5 describes how this design was implemented and explains how it evolved from the initial proposal.

4.1 Prior Work on DISME and Limitations

4.1.1 Modelling in DISME

DISME allows users to define organisational systems through models stored as metadata. The System Specifier supports structural elements, including entity types and properties, while the Action Rules Manager defines the behaviour associated with transactions. The System Executor interprets these models at runtime, following the AOM architecture described in Section 2.2.1.

The Action Rules Engine introduced in [8] extended DISME with business logic. Its formal language is described by an EBNF grammar, and the Blockly editor allows users to construct rules with visual blocks instead of writing conventional source code. The editor restricts how blocks can be combined, while the code generators convert the workspace into the representation used by the platform.

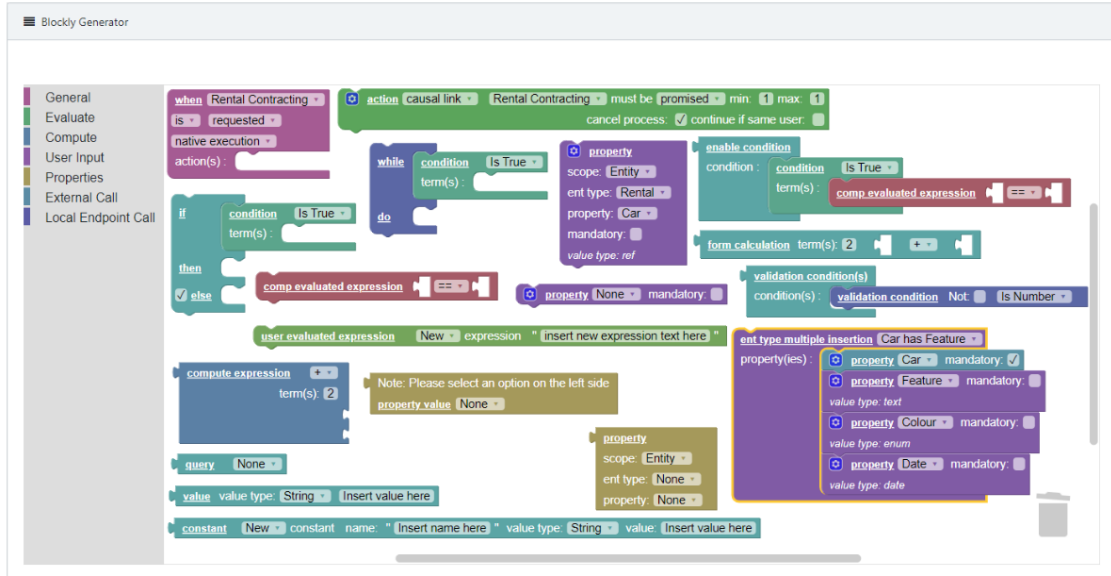


Figure 4.1: The Action Rules Manager interface with the integrated Blockly editor. From [8].

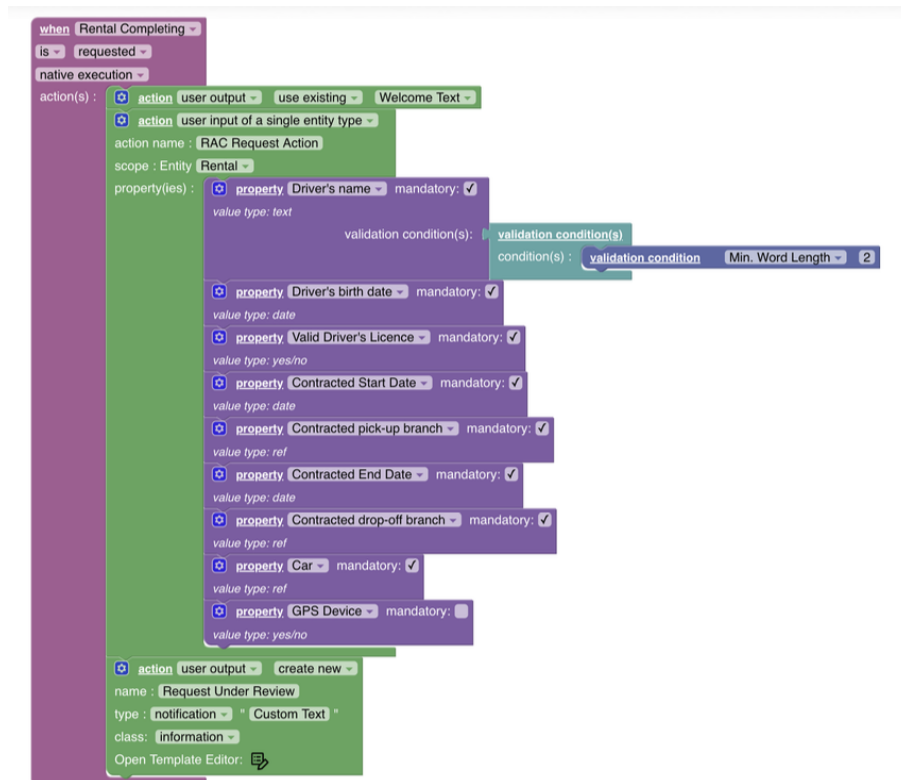


Figure 4.2: Action Rule specification example using Blockly. From [8].

This support removes the need to write the rule in a programming language, but the user still has to identify the required model elements, select the correct blocks, and

organise the control flow. The visual representation changes how the rule is expressed, but it does not define the business process for the user.

4.1.2 The Modelling Barrier

The proposed approach assumes that users can describe the intended system, including its processes, participants, information, and rules. Their experience with DISME may vary. Creating a specification directly through the platform's interfaces requires knowledge of entity types, properties, roles, processes, transactions, queries, and Action Rules, as well as familiarity with the System Specifier, the Action Rules Manager, Blockly, and the transaction states used by the platform.

The difficulty is not limited to programming syntax. For example, a user may know that a review can only be submitted for an assigned article, but still need assistance to represent the required query, condition, form fields, property assignments, and transaction state. Similar problems arise when relationships between entities or the initiator and executor of a transaction need to be defined.

The proposed wizard guides the user through this modelling process by asking natural-language questions about one part of the model at a time. The output is an initial specification that can be reviewed and corrected. The design does not assume that the generated result is correct solely because it has a valid structure.

4.1.3 Design Implications

The Systematic Literature Review (SLR) presented in Chapter 3 found that LLM-generated artefacts still require checks for invalid structure, missing information, and differences from the user's intent [10, 38]. The model can produce a value that follows the expected format while referring to an unavailable element or expressing the wrong business rule. Structural validation and human review therefore address different types of error.

This leads to four design decisions. First, generation is divided into phases so that each part can use the results already confirmed by the user. Second, prompts are built with the model elements and identifiers available at that point in the process. Third, deterministic checks are applied where the platform provides a source of truth, such as supported types, identifiers, references, and Action Rule structures. Fourth, the user reviews the generated content before it is imported or activated.

The design aims to produce an initial specification from the user's business description, reducing the need to create every element manually through the platform's modelling interfaces. Chapter 6 evaluates whether users can understand and correct the generated artefacts and whether those artefacts can be imported and used in the selected workflows.

4.2 System Requirements

This section identifies the users considered by the design and the behaviour expected from the solution. The requirements define the scope of the solution described in Chapter 5.

4.2.1 Stakeholders

The design distinguishes two user roles:

- **Domain user:** a person who understands the business process to be modelled and can describe it in natural language. This user must be able to judge whether the generated entities, processes, and rules express the intended business behaviour. The wizard is designed to support users with different levels of experience with DISME and its modelling concepts.
- **Technical DISME user:** a person familiar with the platform who can provide connection information, complete the administrative configuration, inspect imported Action Rule drafts, and activate them in the Action Rules Manager. The same person may also be the domain user when they have both types of knowledge.

The DISME platform is an external system with constraints that the wizard must respect. These constraints include the available metadata types, the identifiers returned by the API, the transaction states, the supported Action Rule blocks, and the required creation order of dependent artefacts.

4.2.2 Functional Requirements

The proposed solution must:

- FR1** accept an initial description of a business system in natural language and ask follow-up questions about missing details;
- FR2** produce an initial State Model containing entity types, properties, permitted values, and references between entities;
- FR3** produce an initial Process Model containing actor roles, process types, transaction types, initiators, executors, and the transactions that start and end each process;
- FR4** identify and generate the queries required for form fields and Action Rule logic;
- FR5** generate Action Rules for the relevant transaction states using only the structures supported by the implemented intermediate representation;
- FR6** build the context of each phase from information already collected or confirmed, including the identifiers returned by DISME after import;

- FR7** check generated structures and references before allowing their import, and attempt refinement when a generated Action Rule fails the implemented checks;
- FR8** present the generated models, queries, and Action Rules before their corresponding import, and support natural-language corrections for the State Model, Process Model, and Action Rules;
- FR9** present each Action Rule through a short explanation and a list of its actions, conditions, form fields, assignments, and messages when these elements are present;
- FR10** import confirmed State and Process Models and queries through the DISME API;
- FR11** convert validated Action Rules into Blockly XML and, after user confirmation, import them through the draft mechanism of the Action Rules Manager;
- FR12** create the initial forms required by saved Action Rules when the corresponding actions and properties are available in DISME.

4.2.3 Non-functional Requirements and Scope

- NFR1** A blocking structural or referential error detected by the implemented validators must prevent the affected import.
- NFR2** The domain user must be able to perform the main reviews in the wizard without reading raw JSON, Blockly XML, or database records.
- NFR3** Structural artefacts must only be imported after the user confirms them in the wizard.
- NFR4** An imported Action Rule must remain a draft until a user opens and saves it in the Action Rules Manager.
- NFR5** Context passed between phases must use the confirmed session data and, after import, the identifiers returned by DISME.
- NFR6** Repeating an import must reuse an equivalent existing element where this can be established and stop when an incompatible element would otherwise be reused.
- NFR7** The wizard must save its current session state so that an interrupted session can be recovered.
- NFR8** The wizard must use the existing DISME interfaces where they provide the required behaviour, including its RESTful API and Action Rule draft mechanism.

These checks can enforce only the constraints represented in the code and in the information received from DISME. They cannot determine whether a structurally valid

model expresses the organisation's intended process. That decision remains part of the user's review.

The wizard does not create a complete UserApp. User accounts, role assignments, and associations between accounts and domain entities must be configured in DISME. A technical user must also open and save the imported Action Rule drafts before the rules become available to the System Executor.

4.3 Proposed Solution Architecture

4.3.1 Architectural Overview

The proposed solution is a wizard that runs separately from DISME and connects to it through its RESTful API. It produces metadata for the platform rather than source code for a separate application. The architecture includes four components within the wizard and the DISME environment:

- **Conversation and phase orchestration:** collects the description of the intended system, controls the order of the interviews, and presents one modelling topic at a time.
- **Context construction and structured generation:** builds the context from the current session, confirmed models, and imported identifiers, and uses it to generate models, queries, and Action Rules in defined formats.
- **Validation and user review:** checks the generated artefacts, attempts to correct invalid Action Rules, and allows the user to review and confirm the results. Action Rules are also converted into Blockly XML before approval and import.
- **Integration layer:** imports confirmed State and Process Models and queries through the DISME API, and creates or updates Action Rule drafts using the generated Blockly XML.
- **DISME configuration and runtime:** provides the native editors, persists the imported models, activates saved Action Rules, and executes the resulting system after the required administrative configuration.

Generation and review take place across the modelling phases, with imports providing the identifiers needed by later phases.

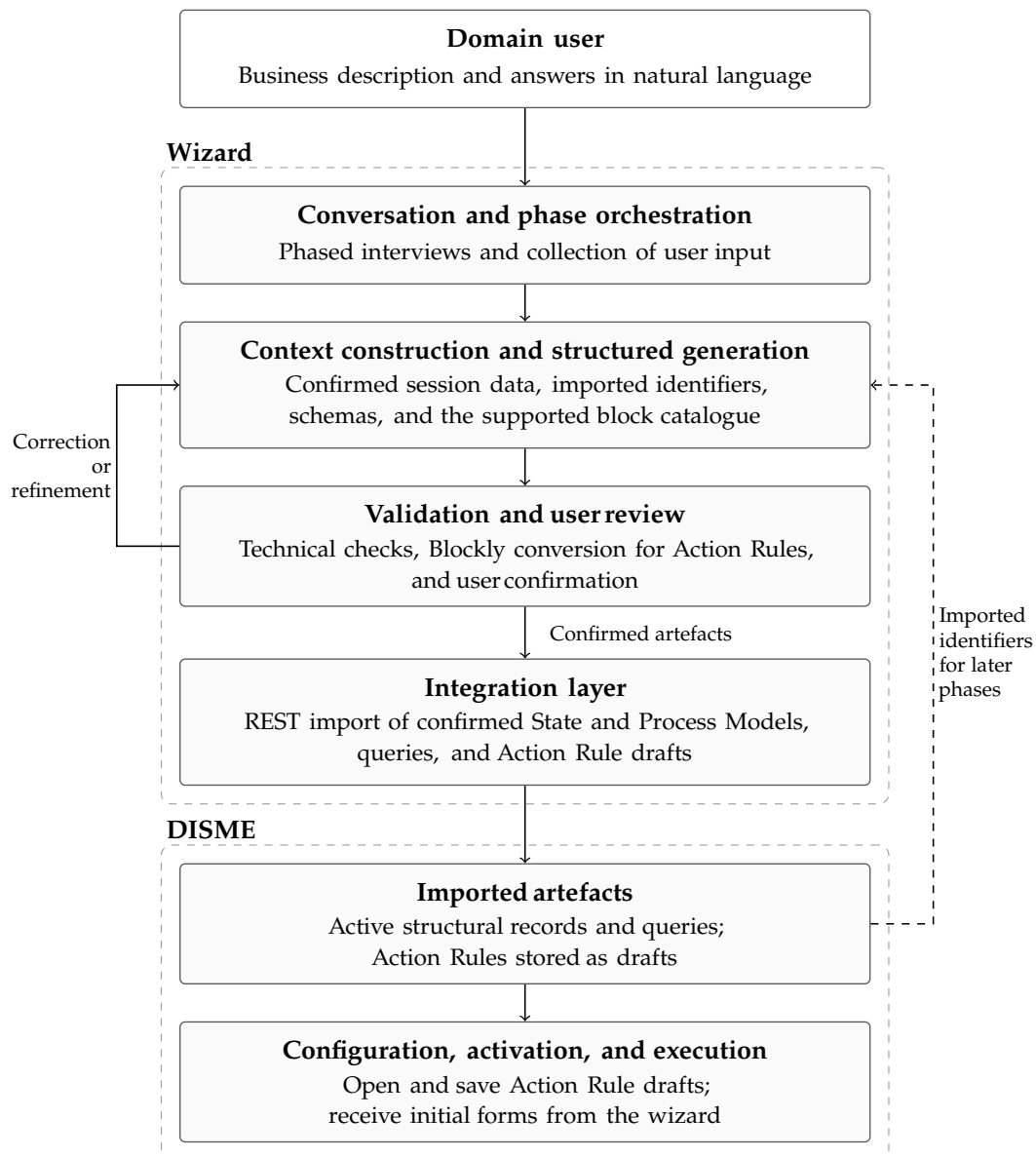


Figure 4.3: Conceptual architecture of the proposed wizard and its integration with DISME.

4.3.2 Phased Generation and Context Construction

The wizard divides the work into phases because later artefacts depend on earlier ones. Phase 0 collects a general description of the organisation, its users, and its main operations. Phase 1 develops this description into the State Model. Phase 2 identifies the roles, processes, and transactions that form the Process Model. These models are reviewed before they are imported because the queries and Action Rules created later require the identifiers assigned by DISME.

Phase 3 defines queries used to filter values shown in forms. Phase 3b handles queries required by Action Rule logic. Phase 4 then defines the behaviour associated with each

relevant transaction. Its outputs include form or edit actions, automatic assignments, conditions, messages, references to confirmed queries, and causal links when the user's description requires them.

Each request receives only the context required for its current task. Before the first import, this context comes from the user's conversations and the models confirmed in previous phases. After the State and Process Models have been imported, the context also contains the real identifiers returned by DISME. The wizard therefore builds query paths and Action Rule references from known elements instead of using unverified identifiers.

The design uses prompts that define the purpose of the interview, the expected output, and the relevant DISME concepts. Structured extraction is separate from the conversational response. In the later phases, JSON Schemas define the required output shape. When generating Action Rules, the model also receives a catalogue based on the supported grammar and block definitions, together with examples of the intermediate representation. Chapter 5 describes these prompts, schemas, and contexts.

4.3.3 Deterministic Validation and User Review

The pipeline applies deterministic checks after extraction and before the affected artefact is imported. For the State and Process Models, the checks cover required fields, supported data types, local identifiers, references between models, transaction states, and dependencies needed by the importer. Query checks cover their target entities, filter properties, paths, operators, parameters, and permitted fixed values within the implemented scope.

Action Rules use a structured intermediate representation rather than direct relational database records. A JSON Schema defines the available node structures. Further validators compare the generated identifiers and values with the imported State and Process Models, the query mappings, and the supported transaction states. When these checks find a correctable generation error, the pipeline can request a new version using the validation feedback. The system makes only a limited number of correction attempts, so some invalid results may remain unresolved. A rule that does not pass the checks cannot be approved or imported.

Deterministic checks do not replace user review. The wizard presents the structural models during their construction and accepts corrections before confirmation. Query specifications are displayed before import. For Action Rules, the final review shows a natural-language explanation and the generated actions. The explanation makes the rule easier to understand, but does not prove that it is correct. The user can approve a rule, signal a problem and request regeneration, or remove a duplicate. The wizard blocks import while a rule remains unreviewed, has a reported problem, or lacks valid XML.

This separation reflects two different questions. The validators check technical aspects of the generated artefacts, such as their structure, supported value types, and references to model elements. The user checks whether it represents the intended business process. An artefact can be imported only after it passes the implemented checks and the user

confirms it.

4.3.4 Integration with DISME

The integration path depends on the artefact. After confirmation and a final preflight, the importer sends roles, processes, transactions, entity types, and properties to the DISME API. These endpoints create active structural records. Unconfirmed versions therefore remain in the wizard rather than being stored temporarily in DISME. The importer stores mappings between the local identifiers used during generation and the identifiers returned by the platform. These mappings are used by queries and Action Rules. Confirmed queries are imported after review and before the Action Rules or forms that depend on them.

Action Rules follow a separate path. The Blockly Bridge loads the block definitions and code generators used by DISME, constructs a headless workspace from the validated intermediate representation, and produces Blockly XML. The importer submits this XML to the existing Action Rule draft endpoint. The draft is not available to the System Executor until a user opens it in the Action Rules Manager, checks the Blockly representation, and saves it as an active rule.

Forms are created after the corresponding drafts have been saved because the required Actions and Action Properties only exist at that point. The importer generates an initial Form.io definition from those records and keeps an existing form when one is already present. The generated form can later be edited through the Forms Manager.

The integration also provides the runtime information required by the selected workflows. This includes the domain entity associated with the current user, the entity from which a transaction started, and values derived from the current date or time. It also completes the execution path for transaction rollback. These extensions allow the generated artefacts to run through the existing DISME components, but they do not remove the administrative tasks stated in Section [4.2.3](#).

IMPLEMENTATION

This chapter presents the implementation of the design described in Chapter 4. The implementation follows the functional and non-functional requirements defined in Section 4.2. It first describes the main components, the models and execution environment, and the workflow of the implemented system. It then follows the phases of the wizard, from the initial business description to the generation of Action Rules. Finally, it explains the integration with DISME, presents a short walkthrough of the complete process, and summarises how the system evolved from the initial proposal.

5.1 System Overview

5.1.1 Architecture and Components

DISME follows an AOM architecture. As a result, the wizard needs to generate metadata that the platform can interpret instead of compiled source code. We implemented the wizard as a Python application with a Streamlit user interface. It runs separately from DISME and manages the conversations with the user, the review of the generated artefacts, and the order in which the phases are executed. The wizard connects to the platform through its RESTful API. Internally, we separated the user interface from the logic of each phase. The phase modules contain the prompts and extraction functions, while a common client layer provides access to the LLMs. The deterministic checks that are part of the current Streamlit workflow are described with the phases in which they are applied.

The wizard uses an importer to send State and Process Models, queries, and Action Rule drafts to DISME. Before Action Rules are imported, the Blockly Bridge loads the block definitions and code generators of DISME, builds the required blocks in a headless workspace, and produces the rule representation and Blockly XML.

During a session, the wizard keeps the current phase, conversations, confirmed artefacts, import results, and identifier mappings in the Streamlit session state. Since the workflow can require several interviews and imports, selected session data are also written to an autosave file after relevant changes and phase transitions. The autosave is written atomically so that an interrupted write does not replace the previous state with an

incomplete file. When the wizard is restarted, it can load the autosave and reconstruct the business draft and identifier mappings needed to continue the workflow. This mechanism implements requirement NFR7 from Chapter 4.

5.1.2 Models and Execution Environment

We first used Claude models provided by Anthropic: we used Claude Haiku for the interviews and Claude Sonnet for extraction through the initial tests. We later added Gemini to the common client layer and tested it in the initial phases. In these tests, Claude and Gemini required fewer corrections to their questions and extracted output than Llama. However, the tests did not use the same cases and conditions, so we do not treat this observation as a controlled comparison.

Both options depended on paid external services. To avoid per-request costs and use self-hosted models, we chose Ollama for the final solution. This choice also allows us to control the inference environment. The supervisor provided access to a workstation with two NVIDIA RTX 5090 Graphics Processing Units (GPUs), where Ollama runs. We use Secure Shell (SSH) to manage the workstation while the wizard sends inference requests to Ollama's API using Hypertext Transfer Protocol (HTTP).

We used the Llama 3.3 70B (llama3.3:70b) model for the interviews and for generating a brief explanation of each Action Rule so that the user could understand it better, as well as for extraction in Phases 0, 1, 2, and 3. Llama performed well in these phases and did not show structural issues until Phases 3b and 4. In these phases, Llama tended to produce structures that were either incomplete or invalid. We therefore used Qwen3-Coder 30B (qwen3-coder:30b) for structured extraction. We made this change because, in our tests, Qwen3-Coder 30B showed a better understanding of the required JSON structures than Llama 3.3 70B and produced better results in these two phases.

Table 5.1 summarises the models, temperatures, and output limits used for the main interview and extraction calls. Extraction calls use a temperature of 0 to reduce variation.

5.1.3 Workflow

The implemented workflow follows these steps:

1. Phase 0 elicits the initial business description and produces a draft that the user can review and correct.
2. Phase 1 extracts entity types and properties. Phase 2 extracts actor roles, process types, transaction types, and process entry and exit information. The user reviews the output during both phases.
3. After this review, the wizard imports the outputs of Phases 1 and 2 into DISME. This import is performed before the remaining phases because queries and Action Rules need to refer to the identifiers assigned by the platform.

Phase and task	Model	Temp.	Max. output (tokens)
Phase 0: interview	Llama 3.3 70B	0.2	400
Phase 0: draft extraction	Llama 3.3 70B	0	2000
Phase 1: entity interview	Llama 3.3 70B	0.2	500
Phase 1: entity extraction	Llama 3.3 70B	0	4000
Phase 2: role interview	Llama 3.3 70B	0.2	500
Phase 2: role extraction	Llama 3.3 70B	0	2000
Phase 2: process interview	Llama 3.3 70B	0.2	500
Phase 2: process extraction	Llama 3.3 70B	0	4000
Phase 3: form query interview	Llama 3.3 70B	0.3	500
Phase 3: form query extraction	Llama 3.3 70B	0	2000
Phase 3b: rule query interview	Llama 3.3 70B	0.3	600
Phase 3b: rule query extraction	Qwen3-Coder 30B	0	6000
Phase 4: rule interview	Llama 3.3 70B	0.2	600
Phase 4: rule extraction	Qwen3-Coder 30B	0	4096

Table 5.1: Model settings for the main interview and extraction calls.

4. Phase 3 defines queries used to filter values in forms. Phase 3b defines queries required by the logic of the Action Rules. Each set of queries is reviewed and imported before the next phase uses it.
5. Phase 4 generates Action Rules for the relevant transaction states. The rules are checked, converted into Blockly XML, and presented for review. Approved rules are imported through the existing draft mechanism of the Action Rules Manager. The user must open and save each draft in DISME. The wizard can then create the forms required by the saved rules that do not already exist.

Structural artefacts are reviewed in the wizard and imported as active records only after confirmation. Action Rules are also reviewed in the wizard but are imported as drafts, which allows the user to inspect their Blockly representation in DISME before saving them.

The interviews follow a common pattern. After the opening message, the prompts ask for one question at a time. The marker [PRONTO] signals the end of an interview or stage and is removed before the response is displayed. A separate extraction call uses the conversation to produce structured output. The wizard presents the generated artefacts for review, with the available correction controls described in the following sections.

5.2 Business Description (Phase 0)

Phase 0 collects an initial description of the business before the wizard asks for model details. Its purpose is to identify the domain, the main users, the problem to address, and the operations that the system needs to support. The wizard uses this result as the starting context for the State and Process Model interviews.

5.2.1 Interview

We used a system prompt to assign the LLM the role of a business analyst and control the interview. The prompt states that the user is the source of information and that the model must not add requirements based on its own knowledge of the domain. It also requires plain language and answers in the same language as the user. When the conversation is in Portuguese, the prompt specifies European Portuguese.

The interview on this Phase 0 consists of two stages. The first asks what the organisation does, who will use the system, and what problem the system should address. The prompt instructs the interviewer to finish this stage only after the user has described both the organisational context and the main problem. The second stage asks about the main operations, the people involved, and the information that the system needs to record. Together, these questions provide a general view of the business. Properties, relationships, and process details are handled in later phases.

The prompt also limits replies to two sentences and asks follow-up questions when an answer is vague. Each request includes the conversation so far and the prompt for the current stage. The interview temperature (Table 5.1) allows some variation in question wording, while the prompt defines the purpose and length of the questions.

At the end of the first stage, the wizard changes the stage index and starts the questions about the main operations. At the end of the second stage, it sends the complete conversation to the extraction step.

5.2.2 Draft and Review

The extraction prompt defines a JSON draft containing the business domain, the conversation language, a classification of the business concepts mentioned by the user, and initial lists of entities and processes. Each entity has a name and a short description. Each process has a name, its main entity, and a short description of its purpose.

The extraction prompt classifies each business concept as a stored record, role, attribute, operation, or uncertain case. Only records and uncertain cases can be proposed as entities, and entity names must use title case. Each proposed entity is then checked against the user's original responses in a separate verification call. Deterministic checks ensure that every candidate is covered and that each quoted excerpt appears in the referenced response. The wizard parses the returned JSON and converts it into a `SpecificationDraft`. The draft does not yet contain detailed properties, relationships, or Action Rules.

The review screen displays the domain, language, entities, and processes identified from the conversation. The user can request corrections in natural language, including the addition or removal of entities. The wizard presents a revised proposal, which replaces the current draft only after the user accepts it. After the user confirms the entity list, the wizard starts Phase 1. The following interviews use the confirmed domain, language, and entity names as context.

For example, in the reduced conference scenario used for evaluation, the user described a system that stores articles and supports their submission and evaluation. Phase 0 proposed *Article* as the stored entity. The people responsible for submitting articles and recording decisions were treated as roles rather than as separate entities. Submission and decision recording were retained as operations for the following phases.

5.3 State and Process Models (Phases 1 and 2)

The result from Phase 0 provides the business context and an initial list of entities and processes, but at this stage, it does not contain the information required by DISME. Phases 1 and 2 develop this draft into the two models that describe what the system stores and how its processes are organised. Phase 1 defines the entity types, their properties, and the relationships between them, while Phase 2 identifies the actor roles, processes, and transactions that act on those entities. The interview and review mechanism is similar to the ones in Phase 0, but the questions are directed at individual parts of the system.

5.3.1 State Model

Phase 1 starts from the entity list confirmed by the user at the end of Phase 0. The wizard handles one entity at a time, instead of trying to define the complete State Model in a single response. This keeps the interview focused on the entity being described and reduces the chance of assigning a property to the wrong entity. It also allows the user to review each result before the wizard continues to the next one.

At the beginning of each interview, the model receives the business domain, the conversation language, the complete list of entities, and the name of the entity currently under discussion. It first asks which information needs to be stored for that entity. As the user answers, the interviewer establishes which properties are mandatory, whether any property accepts more than one value, and which operation creates the entity. When the user mentions a status, type, category, or another field with a fixed set of options, the interviewer asks for the permitted values. It also asks if the entity refers to any of the other entities identified in Phase 0. These instructions are part of the system prompt, but the order of the questions follows the information provided during the conversation.

Once the required topics have been covered, the model returns a short summary and signals completion. The wizard then passes the conversation to the extraction step.

Using the complete interview, the extractor produces JSON containing one `ent_type` object and a collection of `properties`. The entity object includes its local identifier, name, internal name, description, creation transaction, and the flags required by DISME. Each property records its name, type, scope, whether it is mandatory or accepts multiple values, and the information required for fixed options or references. Property types are limited to `text`, `int`, `bool`, `enum`, `double`, `date`, `time`, `prop_ref`, and `file`. The prompt also tells

the model to preserve the property names used by the user and to write the descriptions in the language of the interview.

After the first extraction, a rule-based check looks for missing information such as an entity without enough properties, or a missing creation transaction. If one of these cases is found, the wizard returns to the interview with a question directed at that missing parameter and extracts the entity again. This follow-up can be repeated up to three times before the result is presented for review. The checks applied to the State and Process Models are described further in Section 5.3.3.

Relationships need additional context because an entity may refer to another one that has not yet been interviewed. We handle this with a property of type `prop_ref` that holds the target entity's identifier. Initially, the extractor was limited to entities already processed, so any reference to an entity processed later would go unresolved. The current approach is different: before the interviews begin, every entity in the Phase 0 draft is assigned a local identifier and the complete list is part of each extraction request. Where an entity has been processed, the prompt supplies its properties; those yet to be processed are marked as such. This allows the extractor to record the target identifier immediately, leaving the resolution of the display property until all entities have been defined.

The review screen presents the entity description, the transaction that creates it, and a table of its properties. For each property, the user can see its type, whether it is mandatory, whether it accepts multiple values, and, where applicable, its permitted values or referenced entity. A correction can be written in natural language. The wizard lists the proposed changes, and the user must accept them before they replace the previous result. The confirmed result is then added to the State Model and the wizard moves to the next entity. Once the list has been processed, a final screen with a summary of the result of the model is shown for review.

For example, the confirmed *Review* entity in the CONFERENCE case contains the properties *Score*, *Recommendation*, *Comments*, and *Status*. The same entity refers to *Reviewer* and *Article* through properties of type `prop_ref`. This example shows how information collected in one interview becomes both stored data and links to other parts of the State Model, without requiring the user to define the corresponding DISME records directly.

Figure 5.1 shows the review of a generated State Model before it is confirmed by the user.

5.3.2 Process Model

Once the State Model has been confirmed, Phase 2 describes who takes part in the system and how each entity is managed through transactions. The phase begins with a general interview about actor roles because these roles are needed to identify who requests and who executes each transaction. It then moves through the entities defined in Phase 1 and discusses the process associated with each one.

Fase 1 concluída — Revisão Final

8 entidades, 33 campos extraídos.

- > Article (6 campos)
- > Author (3 campos)
- > Reviewer (3 campos)
- > Review (6 campos)

Evaluation of an article by a reviewer.

Criada por: Submit Review

Campo	Tipo	Obrig.	Editável	Detalhe
Score	Número inteiro	sim	sim	
Status	Escolha de uma opção	sim	sim	Pending, Submitted
Recommendation	Escolha de uma opção	sim	sim	Accept, Reject, Minor revision, Major revis
Comments	Texto	não	sim	
Reviewer	Escolha de informação guardada	sim	não	-> Reviewer (Name)
Article	Escolha de informação guardada	sim	não	-> Article (Title)
- > Assignment (5 campos)
- > Configuration (5 campos)
- > Authorship (2 campos)
- > Conflict Declaration (3 campos)

Correcção global (opcional):

Aqui pode alterar as informações guardadas e os seus campos.

Aplicar correcção global

Continuar para Fase 2

Scientific Conference Management ▶

Figure 5.1: Review of a generated State Model in the wizard.

5.3.2.1 Actor roles

The first interview receives the business domain, the conversation language, and the entity names from the State Model. It asks who will interact with the system and what each person is responsible for, while also keeping in mind differences in access and external people. The interviewer also asks whether the system starts any complete transaction without a person requesting or carrying it out. If so, that transaction uses a *System* role as its executor.

The extractor returns a JSON array of roles with sequential local identifiers, names, and descriptions. A *System* role is added only when the user has described an automatic action. The resulting roles are shown to the user for review and can be corrected before the process interviews begin.

A role and an entity may have the same name without representing the same part of the model. A person is represented as an entity when the system stores information about them, and as a role when they take part in transactions. The State Model describes their data, and the Process Model describes their responsibilities.

5.3.2.2 Processes and transactions

The processes suggested in Phase 0 are not used directly as the final process types. Instead, the wizard follows the entities confirmed in Phase 1 and starts a separate interview for each one. Earlier extractions of the complete Process Model sometimes combined the lifecycles of different entities into one process. Handling them separately was intended to keep the conversation directed at one lifecycle and reduce the chance of combining operations that belong to different entities.

For each entity, the interviewer receives the confirmed roles, all the entity names, and the processes already covered. The conversation begins with the action that starts the process and then establishes who requests it, who carries it out, and what it creates or changes. From there, the interviewer follows the intermediate steps, asks about other operations that can be performed on the entity, and identifies the action that ends the process. It also checks whether any of these steps occurs automatically. Transaction names combine an action verb with the object of the operation, as in *Submit Article* or *Assign Reviewer*. General names such as *Do* or *Complete* are excluded because they do not identify the operation.

The distinction between requesting and executing a transaction is kept in mind during the interview. A request is associated with one or more `initiator_roles`, while `executer_role_id` identifies the role that performs the work. These values may refer to the same role, but the prompt does not assume that they do. When the interview is complete, the extractor receives the conversation together with the available role identifiers and the entity names from Phase 1.

The resulting JSON contains one `process_type` and its `transaction_types`. The process has a local identifier, a name, a description, and a colour selected from a predefined palette. Each transaction has its own local identifier, the roles that request and execute it, a transaction name and request name, and a description. The `result_name` field links a transaction that creates or changes data to an entity from the State Model. Three flags record whether the transaction can start the process, end it, or be activated automatically: `init_proc`, `end_proc`, and `auto_activate`.

Every process must have at least one entry transaction and one terminal transaction, although DISME can represent more than one entry or exit for the same process. A process with a single operation does not need an additional closing transaction, since the same transaction can have both `init_proc` and `end_proc` set to 1. If `auto_activate` is set, the executor must be the *System* role identified earlier. The five transaction states represented in DISME, *Requested*, *Promised*, *Executed*, *Declared*, and *Accepted*, are fixed records already

present in DISME. They are included in the Phase 2 result with their existing identifiers rather than extracted from the conversation.

5.3.2.3 Review

Before moving to the next entity, the review screen presents the process and its transactions in a table. For each transaction, it shows who requests it, who executes it, its result, and whether it starts, ends, or is automatically activated. The user may describe a correction in natural language or confirm the process as it is. After all entities have been covered, the wizard displays the roles, processes, and transactions together in a final review.

During a development test of the broader CONFERENCE case, the first extraction for *Article* grouped submission, reviewer assignment, review, and decision operations in one *Article Management* process. After several corrections during review, the complete Process Model contained five processes: article submission, reviewer assignment, review, decision, and governance. This example shows why user review is still needed, even when the interviews focus on one entity at a time.

5.3.3 Model Validation

The outputs of Phases 1 and 2 are checked by deterministic validators before they are imported into DISME. The same checks are applied to proposed corrections and to the complete models before import. Validation errors prevent an invalid correction from being accepted or disable the import action. Warnings indicate potential issues that may require review, but they do not stop the workflow.

For the State Model, the validator checks the local identifiers and names of entities and properties, the association of each property with an existing entity, and the supported value types. The validator rejects any duplicate identifier for an entity, repeated property names within the same entity, enumerated properties without options, and it also rejects entities that do not have at least one property. For a property of type `prop_ref`, both the target entity and the property used to display the referenced record must exist in the State Model. Warnings are presented when entity descriptions are missing.

For the Process Model, the validator checks the identifiers and names of roles, processes, and transactions. Each transaction must refer to an existing process, a valid executor, and at least one valid initiator. When a transaction has a `result_name`, it must correspond to an entity from Phase 1, and the values of `init_proc`, `end_proc`, and `auto_activate` must be valid. Every process must contain at least one initial and one final transaction. An automatically activated transaction must use the *System* role as its executor. The wizard validates whether the creation transaction stored for each entity exists in the Process Model before importing.

The Phase 2 extractor warns when a process contains a possible invented closing transaction that was not specified by the user. The wizard does not remove it automatically

because a distinct closing step may be valid. Missing process or transaction descriptions are also reported as warnings.

This set of checks covers the structural and referential conditions checked before import. However, they do not guarantee that the imported entities, their corresponding properties, roles or transactions accurately describe the relevant parts of the business domain. The user reviews and confirms that part of the model before import.

5.4 Queries (Phases 3 and 3b)

Queries are defined after the State and Process Models have been confirmed and imported into DISME because they need to identify entities and properties using the identifiers assigned by the platform.

The implementation separates queries according to where their results will be used. Phase 3 defines queries that restrict the values presented in form fields, while Phase 3b defines the queries required by the logic of the Action Rules. Both are represented through the query mechanism of DISME, but form queries are associated with a property that refers to another entity, and Action Rule queries are used to retrieve records while a transaction is being executed.

5.4.1 Form Queries

When a form contains a property of type `prop_ref`, the corresponding field allows the user to select a record from another entity. Without a query, the field may present every available record of that entity. Phase 3 determines whether this list needs to be restricted and, if so, which conditions should be applied.

At the beginning of the phase, the wizard identifies the reference properties in the State Model and considers them one at a time. For each property, the interviewer asks if the list presented in the form needs a filter. If the user requests one, the conversation confirms the property to use and whether its value is fixed or supplied by the user at runtime. More than one condition can be associated with the same query. When the State Model contains no reference properties, the wizard skips this phase and continues to Phase 3b.

The interview prompt receives a context built from the confirmed State Model and the identifiers returned by DISME. This context contains the properties that can be used as filters and their available paths. These paths are prepared by the wizard before the request is sent to the LLM, and the extraction prompt instructs the model to copy one of them rather than produce new identifiers. The construction and validation of these paths are described in Section 5.4.3.

The interview ends once all reference properties have been covered. The output limit for its replies is listed in Table 5.1.

The extractor returns a JSON array with the confirmed queries. Each query records the reference property for which it was created, the entity to be queried, its name, and the conditions to apply. A condition contains the selected property and path, an operator, and either a fixed value or a parameter whose value is supplied at runtime. The supported operators include equality, inequality, partial text matching, and numeric or date comparisons. If the user has not requested a filter, the extractor returns an empty array.

Before import, the review screen explains each query in plain language and displays its properties, operators, and values. Fixed values are shown directly, while runtime parameters are identified as values to be chosen by the user. After confirmation, the importer creates the queries in DISME and stores the identifiers returned by the platform. When the forms are created after rule activation, queries without runtime parameters can be associated automatically with their reference fields. A query with runtime parameters requires a value obtained during the execution of the Action Rule and cannot be attached in the same way to a new entity form.

For instance, the *Assignment* entity in the CONFERENCE case has an *Article* reference property. The user requested that this field present only articles whose *Status* was *Submitted*. The resulting query uses *Article* as its target entity and applies a fixed equality condition to the *Status* property. The query is then associated with the *Article* field when the corresponding form is generated.

5.4.2 Action Rule Queries

Phase 3b deals with the records that an Action Rule may need while a transaction is being executed. For example, a rule may need to check an existing related record or retrieve a collection for later processing. These queries are defined before the rules themselves so that Phase 4 can use the query identifiers returned by DISME.

The interviewer gets the transaction types defined in Phase 2, the State Model, form queries from Phase 3 and the identifiers put in place at import. The model considers each transaction separately and asks whether its logic requires any entity records to be retrieved. When the user confirms that a lookup is needed, the following questions identify the target entity and the properties by which it should be filtered. A transaction may require several queries, and the prompt instructs the interviewer to finish only after all transactions have been considered.

The extraction uses a context window of 16384 tokens to include the transaction and entity context together with the complete interview. The output limit in Table 5.1 allows it to return several query definitions with their filters. The output follows the JSON Schema described in Section 5.4.3. Query names are produced without spaces so that they can be used as identifiers in DISME.

Before import, the generated query specifications are normalised as described in Section 5.4.3.

We discovered during development that a result could be structurally valid and still

omit a query confirmed during a long interview. After the first extraction, the wizard therefore performs an omission audit and checks the parts of the conversation in which the user confirmed that a lookup was needed. Any additional queries pass through the same normalisation process before they are merged with the initial result, and equivalent or partially duplicate definitions are removed.

The review screen shows each query together with its target entity, properties, operators, and values. Fixed values are presented in their original form and runtime parameters are marked as values obtained from the execution context. The user has the option of running the completeness check once more before import. In the event the wizard does not encounter a logic query, the wizard enters Phase 4 with the form queries alone. But if there are any, they have to be imported first to make their identifiers ready for rule generation.

For instance, the *Submit Review* transaction may use a query such as `AssignmentsForArticleAndReviewerWithPendingStatus`. It retrieves records of *Assignment* using the article and reviewer supplied during execution, together with the fixed condition *Status* equal to *Pending*. Phase 4 can then use this query to check whether the reviewer has a pending assignment for the article before processing the review.

5.4.3 Query Paths and Validation

Each query filter must identify both the property used in the condition and the path needed to reach it. The wizard represents this path as a sequence of segments, each containing a property identifier, an entity type identifier, and a direction. A property of the queried entity needs only a final segment. If the property belongs to a related entity, the path first follows the reference and then identifies the property used in the condition. These identifiers come from the mappings returned when the State Model is imported into DISME.

Before each query interview, the wizard uses the confirmed State Model to determine which properties can be used as filters. In Phase 3, these may be properties of the entity shown in the form or properties reached by following one reference to another entity. Phase 3b currently uses only properties that belong directly to the entity being queried. The LLM receives the available paths and their identifiers, so it selects an existing path instead of creating one. When the query is imported, the wizard converts the selected path into a `pathKey`, which is the text representation used by the DISME Query Builder to identify that path.

For form queries, the importer checks that a path is present and that each of its segments contains the required identifiers and a supported direction. When a filter uses a fixed value from an enumerated property, the importer maps its label to the corresponding identifier stored by DISME. If the path is incomplete or the enumerated value does not exist, the query is not imported.

The expected output structure is defined in the Phase 3b module as a domain-independent JSON Schema. It specifies the fields and value types required for a query, but does not contain entities, properties, or values from a particular system. The wizard sends this schema to the LLM together with the extraction request. The LLM then fills the structure with information from the interview, the State Model, and the identifiers returned by DISME.

Before import, Phase 3b compares the generated result with the State Model. This check confirms that the target entity and filter property exist, that the direct path connects them, and that the selected operator is supported. During this process, the wizard also normalises query names, matches enumerated values regardless of capitalisation, and keeps values supplied at runtime as parameters. Some fixed filters depend on the date or time at which the query is executed rather than on a value given during the interview. For these cases, the wizard accepts `CURRENT_DATE`, `CURRENT_DATETIME`, and `CURRENT_TIME`, but only when the selected property has a compatible type. Queries recovered by the completeness checks pass through the same process.

For Phase 3b, these checks determine whether a query has a structure that can be imported and whether its references exist in the confirmed model. For form queries from Phase 3, the importer checks the structure of each path and the fixed values used with enumerated properties, but it does not compare every identifier with the State Model. Neither process determines whether a query expresses every condition intended by the user, which is why the result is still presented for review and compared with the interview. The current path generation also has a limited scope: Phase 3 prepares direct and single forward paths, while Phase 3b uses direct paths. Although the shared path format and the importer recognise reverse segments, the current interviews do not generate them.

5.5 Action Rules (Phase 4)

After the State and Process Models and the required queries have been imported, Phase 4 defines what should happen during each transaction. The wizard uses the user's answers and the identifiers returned by DISME to generate structured Action Rules. These rules are checked and converted into Blockly blocks before they are imported.

5.5.1 Rule Interview and Generation

The wizard considers one transaction at a time. This keeps the conversation directed at the behaviour of the current transaction and allows the user to review its rules before moving to the next one. If a transaction does not require an Action Rule, the user can skip it and continue with the remaining transactions.

Before each interview, the wizard builds a context from the confirmed State and Process Models and the mappings obtained during import. This context contains the transactions, entity types, properties, permitted values, transaction states, and queries available to the

rule. It uses the identifiers returned by DISME, since the generated rule must refer to records that already exist in the platform. Rules confirmed for earlier transactions also provide the mandatory setting of fields when the user states that a form should reuse them.

The system prompt describes an Action Rule as behaviour associated with a transaction state, while keeping the questions in business terms. It specifies European Portuguese and treats the user as the source of all requirements.

The prompt instructs the interviewer to first ask what type of form the transaction uses and which fields the user must fill in. Depending on the transaction, the form may be used to create a new record, edit an existing record, or link multiple existing records. After identifying the fields, the interviewer asks about:

- filters for reference fields, default values, and field validation;
- values assigned automatically by the system;
- conditions that may stop or change the Action Rule's behaviour;
- transactions that should start after the current transaction is completed;
- actions that need to be repeated;
- messages shown to the user.

Questions about calculations and values reused between actions are included when the transaction requires them. The interview distinguishes between data entered by the user and actions performed automatically by the system.

During development, the local interview model sometimes returned [PRONT0] too early or repeated a question before covering the remaining topics. The wizard checks whether the main questions have been asked whenever the model attempts to finish or repeats an earlier question. If a topic is missing, the wizard supplies the next question and continues the interview. This check controls the progress of the conversation, while the user's answers continue to provide the content of the rule.

Following the interview, Qwen3-Coder 30B reads the conversation and converts it into a structured rule definition. The first implementation asked for the entire rule definition for the *Executed* state in one extraction. Testing did produce valid structures for the conversations, but many responses omitted user-confirmed actions (e.g., auto assignments or blocking conditions). Since the remainder of the structure was still valid, a structural validation process could not determine if parts of the conversation were excluded.

For this reason, the current implementation divides the extraction for the *Executed* state into four checkpoints:

1. **Form:** generates the action that creates or edits an entity through a form, together with its field settings.

2. **Automatic assignments:** generates values assigned without user input.
3. **Conditions and subsequent actions:** generates conditions, transactions started by the rule, and repeated actions.
4. **Messages:** generates messages shown to the user.

Each checkpoint can return no action when the conversation contains no corresponding requirement. The resulting actions are then combined into a single rule, and duplicate actions generated by more than one checkpoint are removed.

Before the generated parts are combined, the wizard checks them against the information stated in the conversation. These checks and the correction process are described in Section 5.5.4.

The *Accepted* state is handled separately because most of the collected behaviour belongs to *Executed*. The extraction prompt instructs the model to return an Action Rule for *Accepted* only when the conversation refers explicitly to behaviour that should occur after acceptance. When rules are produced for both states, form and edit actions are removed from *Accepted*, as are actions that repeat what was already generated for *Executed*.

Once the generation is complete, the wizard displays the result of the current transaction with a short explanation in natural language. The review identifies the status of the transaction as well as the generated actions, such as form fields, automatic assignments, conditions, messages, and links to other transactions. At this point, the user is able to modify the form name, the fields associated with it, and the mandatory status of each field directly, ask for a new round of generation in natural language, accept the results of the last generation, or discard the generated result and continue to the next transaction. After all transactions have been considered, a final review groups the rules according to whether they are still pending, approved, or marked for correction before import.

5.5.2 Rule Representation and Blocks Catalogue

To connect the output of the LLM to Blockly, we defined a JSON IR for Action Rules. This IR follows the existing block tree used by DISME, and the LLM produces it instead of generating Blockly XML directly. We derived its structure by examining the block definitions and the Blockly XML of Action Rules already available in DISME. The IR stores all the information required to reproduce each block, but the creation of the block connections and the serialisation of the entire workspace as XML is still handled by Blockly. This means that the wizard can limit, validate and modify the rule before the Blockly blocks and XML are produced.

The rules begin with a *when_is_do* block which describes the transaction as well as the state at which the rule is executed. The remaining blocks represent the actions in their execution order. Each block includes both its type and optional items such as fields, values, statements, or a mutation. In general, fields contain identifiers and fixed option choices, values provide links to one single input, statements comprise lists of blocks in

some order, and mutations provide descriptions of changes to the shape of a block. The resulting structure is close to the Blockly representation while remaining suitable for the checks and corrections performed by the Python pipeline.

The following shortened example is taken from the rule that marks a request as handled. It represents the automatic assignment of the value *Handled* to the property *Status*.

```
{
  "type": "when_is_do",
  "fields": {
    "when_is_do_transaction_type": "15",
    "action_rule_type": "ACT",
    "when_is_do_t_state": "3",
    "action_rule_execution_type": "NATIVE_EXECUTION"
  },
  "statements": {
    "actions": [{
      "type": "action",
      "fields": {"action_dropdown": "ASSIGN_EXPRESSION"},
      "values": {
        "assign_expression_first_input": {
          "type": "property_single",
          "fields": {
            "property_output_ent_type": "6",
            "property_output": "23",
            "property_single_entity_scope": "CURRENT_PROCESS"
          }
        },
        "assign_expression_second_input": {
          "type": "property_value",
          "fields": {"property_values": "11"}
        }
      }
    }]
  }
}
```

At the root, `fields` identifies transaction 15 and state 3, which correspond to *Mark Request as Handled* and *Executed* in this system. The `actions` statement contains the blocks in their execution order. Within the assignment block, `values` connects its two inputs: the first identifies property 23, *Status*, of entity 6, *Request*, and the second identifies the permitted value 11, *Handled*. These identifiers are returned by DISME during import and therefore vary between systems.

The conversion of this representation into the formats used by DISME is described in Section 5.5.5.

In the first version of the extraction prompt, the available actions were described through a small set of formats and examples. These examples showed the model how to reproduce the corresponding structures, but provided little information about when the other blocks available in DISME should be used. As such, we created a blocks catalogue based upon the documented EBNF grammar and the block definitions of the platform. There are 28 entries within our implementation of this catalogue that cover root block, actions, control flows, conditions, values, context variables and form field options.

Every catalogue entry relates the name used in the grammar to its type as it will appear in the IR, with description, guidance about where this block should be used, and a simple example. For instance, the catalogue distinguishes between creating and editing an entity, assigning a value without user input, starting another transaction through a causal link, and stopping an operation through a condition and a rollback. The full catalogue is included in each extraction request, so this allows the model to choose which block to generate according to its purpose, rather than solely based on one of the set number of fixed examples contained within the prompt.

The catalogue explains what the blocks mean and how to use them. However, it does not validate that a generated JSON is in an acceptable format, or that a given identifier exists within DISME. As such the JSON Schema that has been defined within Section 5.5.3 defines what structures can be built. The checks described in Section 5.5.4 check the generated references against the imported records in DISME.

5.5.3 Schema-Constrained Generation

Once the IR had been defined, the next challenge was to make the extractor follow its structure. For each generation checkpoint, the extractor (Qwen3-Coder 30B) receives an interview, a context created by the imported models, a catalogue of blocks, the format templates, and few-shot examples. The context contains the identifiers that may be used in the rule, while all other components of the prompt define how those elements can be represented and combined.

The first versions of the prompt contained only simple examples, such as the creation of an entity and the start of another transaction through a causal link. Once additional rules were implemented to allow more complexity, we analysed existing Action Rules from the municipal system (`Create Citizen` and `Create Hearing`) that had been previously

modelled as Blockly XML in DISME, but we did not utilise this XML directly with the prompt. Instead we used it to determine how the individual blocks were organised and then represented those same block patterns in our IR. More examples have been included since new structural elements have been developed.

The current prompt contains 19 labelled few-shot examples in addition to the format templates. The first examples cover individual actions, while the later ones show combinations involving conditions, queries, loops, field validation, default values, and context variables. They show different construction patterns rather than every possible rule. The catalogue tells the model what each block represents and under which circumstances they should be used, while the examples show how to connect multiple blocks together in the same IR.

Prompt instructions and examples did not prevent the model from producing incomplete or malformed structures. We therefore defined a JSON Schema for the IR and pass it to Ollama with each extraction request. The schema requires the root `when_is_do` structure, describes the attributes of each supported block, and defines the formats those attributes may use. Identifiers continue to be represented as strings within the schema, since the identifier's existence cannot be determined by its type.

The schema is organised into three coverage levels:

1. The first level includes actions such as forms, edits, assignments, links, and messages.
2. The second level adds conditions, loops, queries, and context variables.
3. The third level adds field options such as validation, default values, calculations, and queries used to populate dropdowns.

The final configuration uses the third level.

Control blocks introduced another problem because an Action Rule may contain actions inside a condition or a loop. A recursive schema was not stable with the Ollama configuration used during development, so we represented the permitted nesting through a fixed set of levels. This covers the nested structures used by the supported examples without requiring a schema that refers back to itself. The first complete version also repeated the same definitions each time a term, condition, or field appeared and produced a schema larger than 2 MB, which Ollama rejected. We moved these shared definitions to `$defs` and reused them through `$ref`, reducing the schema without changing the IR that the model must produce.

The extractor uses a context window of 32768 tokens to accommodate the interview, the imported model information, the catalogue, and the examples. The output limit in Table 5.1 allows for rules with nested blocks. If the Ollama endpoint rejects the request because the schema format is unsupported, the adapter repeats the request without the `format` field. The parsing and validation steps are still applied to the returned result.

5.5.4 Rule Validation

The JSON Schema defines the structure of the IR, but it does not check whether the identifiers exist in the current system or whether the rule includes everything the user requested. Therefore, after the JSON has been parsed, the pipeline performs a second group of checks before it passes the rule to the Blockly Bridge.

1. **Normalisation of values and references.**

During the final simulation with the simple system, for example, the extractor represented *Pending* as text even though *Status* was an enumerated property. The State Model and the import mappings identified one permitted value with that name, so the pipeline replaced the text with its DISME identifier. This step can also modify the representation of an entity reference as well as the source of a query parameter where one mapping is found.

2. **Reference checks.** The validator then takes the identifiers from the State Model, the Process Model, the queries, and the values imported, and walks through each represented block in the IR (including those within conditions or loops), checking if the transaction state is the one requested for that generation. It also checks whether the referenced transactions, entities, properties, queries, filters, and permitted values exist. Where these elements are connected, the validator confirms, for example, that a property belongs to the stated entity, a permitted value belongs to the destination property, a reference points to the expected entity, and a filter belongs to the selected query.
3. **Correction attempts.** If the response is not valid JSON or fails one of these checks, the pipeline returns the generated output and the identified problems to Qwen3-Coder 30B. The model is asked to return the complete corrected IR and may be called two more times after the first attempt. If all three calls are unsuccessful at generating a valid result the rule fails to make it through to the Blockly Bridge.
4. **Missing actions.** During testing, we also found that an IR could pass these checks while omitting an action confirmed during the interview. Structural and referential tests alone cannot find omissions of actions because they do not check for content within the rules, only structure. Programmed checks look for explicit field names and action patterns in the user's answers and check whether the corresponding elements are present in the generated IR. These checks cover the fields included in a form, automatic assignments, blocking conditions, messages, loops, and transactions that should start automatically. When something has been left out, the wizard asks the extractor to regenerate that checkpoint with the missing content. If the regenerated checkpoint still omits required content, the wizard rejects the rule.

5. **Final validation.** After the checkpoints have been combined, duplicate actions and actions that the user explicitly rejected are removed, and the resulting rule is validated again.

Passing validation means that no errors were found by the implemented checks. These checks do not prove that the rule expresses the intended business requirements. A rule proceeds to the Blockly Bridge only after passing the validations. The user reviews the result before approving it for import.

5.5.5 Blockly Bridge

Once a rule has passed the previous checks, the Blockly Bridge converts its IR into the formats used by the Action Rules Manager. We created this component to reuse the Action Rule implementation already available in DISME instead of reproducing the same logic in Python. Before this work, DISME already defined its blocks in `DISMECustomBlocks.js` and their code generators in `DISMECustomBlocksCode.js`. Since these files and Blockly use JavaScript, we implemented the conversion in Node.js and called it through a small Python wrapper.

The Python component sends the IR together with the entities, properties, transactions, and queries of the current system. At this point, their local identifiers have already been replaced by the values returned by the DISME API. The Node.js component uses this context to populate the options required by the blocks and then loads the block definitions and code generators from the configured DISME installation. It runs Blockly in a headless workspace, supported by a minimal document environment since the conversion does not need access to the visual editor.

Starting with the root `when_is_do` block, the bridge follows the IR recursively. For each element, it creates the corresponding Blockly block, applies its mutation, fills its fields, and connects the blocks contained in its value and statement inputs. Mutations are applied before the fields because they determine the shape of a block and which inputs are available. Statement blocks are connected in the order in which they appear in the IR, thereby preserving the sequence of the actions.

We first assumed it would simply be enough for our program to load all the definitions and populate their respective fields so as to replicate the behaviour of the visual editor. However, during testing, we found that some DISME blocks depend on mutations and change events triggered by the user while working in the visual editor. For instance, selecting a query or adding entity details may not produce the required child blocks when the workspace is built programmatically. Therefore, the bridge creates these blocks and then fills their options using the imported context. The bridge also removes unattached blocks that can be created by some mutations because, if they remain in the workspace, the fragments they generate may be included in the rule.

Once the workspace has been built, the code generators in `DISMECustomBlocksCode.js` generate the JSON representation of the rule and then Blockly serialises the same

workspace to XML. The wizard uses the JSON representation to show the rules that were generated for each action during review and retains the XML for the later import as a draft. If there is a failure converting to XML, the wizard records the errors and does not generate an XML result. A rule can only be approved and imported when a valid Blockly XML is available.

5.5.6 Rule Example

The rule for *Submit Review*, from the CONFERENCE system, brings together several parts of the Phase 4 pipeline. During the interview, the user states that the transaction must present a form for the score, recommendation, and comments, associate the new review with the article and reviewer from the selected assignment, set the review state to *Submitted*, and change the assignment state to *Completed*. The score must also be between 1 and 5. This information is divided between the four generation checkpoints and then combined into one rule for the *Executed* state.

The following outline omits the identifiers and the fields used only to reconstruct the blocks. It shows the order of the actions in the resulting IR.

```
when_is_do(Submit Review, Executed)
  user_input(Score [1..5], Recommendation, Comments)
  assign(Reviewer, origin.Assignment.Reviewer)
  assign(Article, origin.Assignment.Article)
  assign(Review.Status, Submitted)
  update(origin.Assignment, Status = Completed)
  message("Review submitted successfully.")
```

The form action and its validation are generated by the first checkpoint, while the automatic assignments are generated by the second. The remaining update is included in the combined rule, while the confirmation message is generated by the fourth checkpoint. The outputs of all four checkpoints are then combined. At this stage, the wizard uses the mappings obtained from DISME to associate the properties, references, and permitted values with their corresponding records.

Figure 5.2 shows how the combined rule is presented to the user before approval.

The score validation uses separate comparisons to require a value greater than or equal to 1 and less than or equal to 5. This is necessary because the `belongsRange` condition available in DISME excludes the two limits and would therefore reject the scores 1 and 5.

The rule also needs to identify the assignment from which the review was started. When the reviewer starts the transaction from a row in the UserApp, the selected assignment is passed as `origin_entity_id`. This allows the rule to obtain the corresponding article and reviewer and to update the exact assignment instead of selecting one without enough context.

Fase 4 — Submit Review (6/13)

Explicação da regra gerada:

When Submit Review reaches the Executed state, the wizard asks for the score, recommendation, and optional comments. It links the new Review to the Reviewer and Article from the Assignment, marks the Review as Submitted and the Assignment as Completed, and displays a confirmation.

Quando 'Submit Review' chega a Executada:

- Formulário (criar): **Submit Review [3] — Review Form**
- Score *
- Recommendation *
- Comments
- Atribuição automática: **Reviewer** = Assignment.Reviewer
- Atribuição automática: **Article** = Assignment.Article
- Atribuição automática: **Status** = "Submitted"
- Atribuição automática: **Status** = "Completed"
- Mostra uma notificação: "Review submitted successfully."

> Editar campos manualmente

Pedir alteração (ou deixar em branco para confirmar):

Saltar Aplicar alteração Confirmar e continuar

Figure 5.2: Review of the generated rule for *Submit Review* in the wizard.

After validation, the Blockly Bridge creates the form, validation, assignment, reference, update, and message blocks in a single workspace. The code generators already present in DISME then produce the rule representation, while Blockly serialises the workspace as XML. The changes required in DISME to carry the selected assignment through the execution context are described in Section 5.6.5. The execution of this rule is described in Section 5.7.

5.6 DISME Integration

Integration with DISME takes place at different points in the workflow rather than in a single final step. The confirmed State and Process Models are imported after Phase 2 because the remaining phases need the identifiers assigned by the platform. Queries are then imported as they are confirmed, while the Action Rules are imported as drafts after Phase 4. This sequence allows each artefact to refer to records that already exist in DISME.

5.6.1 Model Import

The structural endpoints of the current DISME API create roles, processes, transactions, entities, and properties as active records and do not provide a common draft state for them. Their review therefore takes place in the wizard before import. The draft mechanism already available in DISME is used later only for Action Rules.

5.6.1.1 Import order and record reuse

After the State and Process Models are reviewed and confirmed by the end user, the wizard starts the import, and the importer authenticates with the DISME API. In addition, before the importer creates any records, it checks whether there is an existing transaction with the same name associated with a different process. If this situation occurs, the importer stops the import because reusing that record could cause the generated model to be incorrectly connected to the other process.

The order of the import follows the dependencies between the records. Roles are created or identified first because transactions refer to their initiator and executor roles. Process types are handled next, followed by transaction types, entity types, and properties. Transaction states are not created by the importer because the states used by the generated Process Model already exist in DISME. By following this order, each request can use the identifiers returned for the records on which it depends.

The importer does not assume that the configured DISME instance is empty. Before creating a record, it searches for an existing one with the same identifying fields and reuses it when possible. When a duplicate conflict occurs, the importer searches again for the record and uses the existing one. This way, if an error occurs during the process, such as losing the connection to DISME, the import can be run again without creating multiple versions of the records that have already been processed.

5.6.1.2 Identifier mappings and references

Each time a record is created or a previously created one is used, an association is created between the locally assigned identifier used during generation and its matching identifier in DISME. Once all structural imports have been completed, the wizard retrieves data from the platform regarding the processes, transactions, entities, and properties and builds these mappings again. The mappings that the wizard creates are then stored in the current session, where they can be used to build and import queries and to create references within Action Rules.

Properties that reference another entity require an additional step during the import. To accomplish this, the importer first creates the properties, along with the permitted values for enumerated properties. Once the target properties are available, the property used to display the referenced entity is resolved, and the original reference is updated

with the corresponding DISME identifier. This prevents a reference from being created before its target property has finished being imported.

5.6.1.3 Process Details

After importing the structural records, the importer also configures the Process Details that DISME uses to identify process instances in the dashboard. For each process, the wizard first looks for properties of an entity created by one of its initial transactions. If no match is found, it considers the remaining transactions and the names of the process and entities. These properties are then compared with those that DISME permits for the process. When no preferred property remains after this comparison, the wizard uses the properties returned by the platform. The selection is limited to four properties to keep the displayed details readable, and each property is associated with the process through the API.

When a valid Process Details configuration already exists, the importer keeps it without changes. If the configuration contains a property that is no longer eligible, the import is stopped so that it can be reviewed in DISME. A process may also have no eligible properties, particularly when it only modifies entities created in another process. In this case, the wizard leaves the process without Process Details and reports a warning.

5.6.2 Query Import

The queries cannot be left until the end of the import process as they rely on the creation of entities and properties in DISME, while Action Rules that are created afterwards might also be dependent upon their own queries. For this reason, the wizard imports each group when it becomes necessary. Form queries are imported after they have been reviewed during Phase 3. Action Rule queries are imported after Phase 3b has been completed and just before the rules are generated.

Both groups are handled by the same importer. The confirmed query specification is converted into the structure used by DISME to represent a query, including the entity to be queried, the selected property paths, the returned properties, and the filter conditions. Since these paths are built using the mappings obtained after the Model Import, their entity and property identifiers already correspond to records in DISME. Before sending the query to the platform, the importer resolves the identifiers of all fixed enumerated values referenced by the query, as described in [Section 5.4.3](#).

Before creating a query, the importer searches for one with the same name. If it exists, it reuses it only when the target entity and filter definitions match. If the target entity or filter definitions differ, the importer stops the import to avoid linking artefacts to the wrong query.

After creating or reusing a query, the importer returns its DISME identifier and the identifiers of its parameter filters. The wizard stores these mappings in the session so

that form fields and Action Rules can reference the queries and rules can supply runtime parameter values.

If the Phase 3b completeness check changes the queries and they are imported again, the wizard keeps the form query mappings and replaces the Action Rule query mappings. Mappings for queries removed during review are no longer available to Phase 4.

5.6.3 Blockly XML and Action Rule Drafts

After all the generated rules have been reviewed and approved, the wizard checks that each one contains well-formed Blockly XML before starting the import. In particular, the wizard does not create the Active Rule or add its conditions and actions to the relational tables by itself. Instead, it uses the draft mechanism already available in the Action Rules Manager. This leaves the creation of the active rule within DISME and allows the generated blocks to be opened in the same editor used for manually created rules.

The importer gives each draft a name based on its transaction and transaction state, since the same transaction may have separate rules for the *Executed* and *Accepted* states. It sends the name and Blockly XML to the `/auth/action-rule-drafts` endpoint. The IR and the JSON representation used for review are not used to create active records at this stage.

Draft names must be unique within each language. To support repeated imports, the importer creates a draft if none exists with that name, keeps it if the XML is unchanged, and updates it if the XML differs. It also skips local rules with identical logic and reports an error if different local rules would receive the same draft name.

The draft that was imported into the system is still unavailable to the System Executor. Therefore, a user needs to open this imported draft inside the Action Rules Manager, where DISME loads the saved XML file and recreates the Blockly work area. Figures 5.3 and 5.4 show the imported *Submit Review* draft in two parts after it has been opened in the editor. The user can then inspect or edit the blocks before saving the rule.

When the user saves the rule, the code generators already present in DISME generate its JSON representation from the workspace. The platform then creates or replaces the active Action Rule and stores its actions, conditions, and related records. Only after this step can the System Executor interpret the rule at runtime.

5.6.4 Form Creation

Phase 4 identifies actions that require users to enter or edit data. Their forms can only be created after the user opens and saves the drafts as active rules in the Action Rules Manager, because each Form depends on the resulting Action and Action Properties. Form creation is therefore an additional step at the end of the wizard. If the importer cannot find the active actions, the wizard asks the user to save the drafts in DISME before trying again.

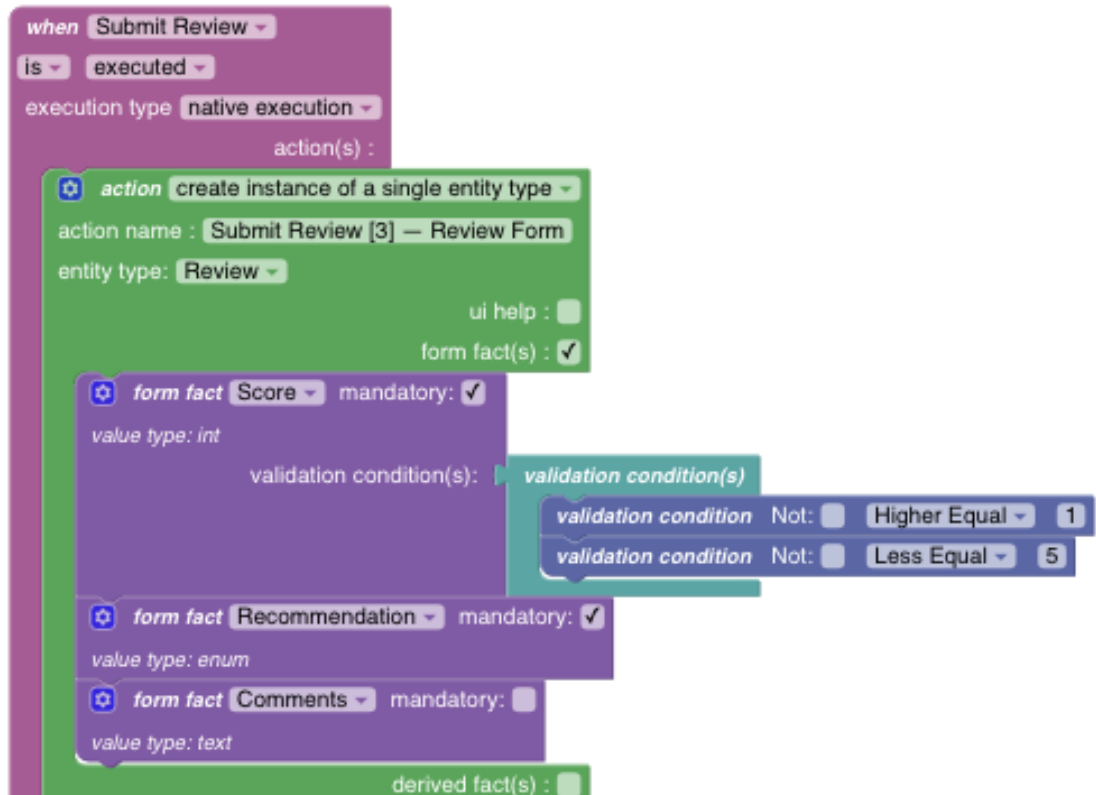


Figure 5.3: Imported draft of *Submit Review*: form fields and score validation in the DISME Action Rules Manager.

The importer retrieves the active Action Rules, actions with form fields, and existing Forms in DISME, limiting the search to transactions imported for the current model. This prevents Form creation for other systems. It considers actions that request user input or edit an entity instance. If an action already has a Form, the importer keeps it unchanged when the import is repeated, preserving any user edits.

For each remaining action, the importer retrieves the Action Properties created when the rule was saved. It generates the Form from these records without another LLM call. The fields are based on the actions and properties confirmed in Phase 4. The JSON generated for DISME defines the field order, default values, tooltips, editability, and support for multiple values.

The type of each property determines the component used in the Form, as shown in Table 5.2.

When an action accepts multiple instances of the same entity, its fields are grouped in a data grid. When the user needs to select existing entity instances, the importer creates a selection field instead of separate fields for their properties.

Some validation information is also transferred to the Form. Required fields and minimum or maximum character and word limits are converted into Form.io validation

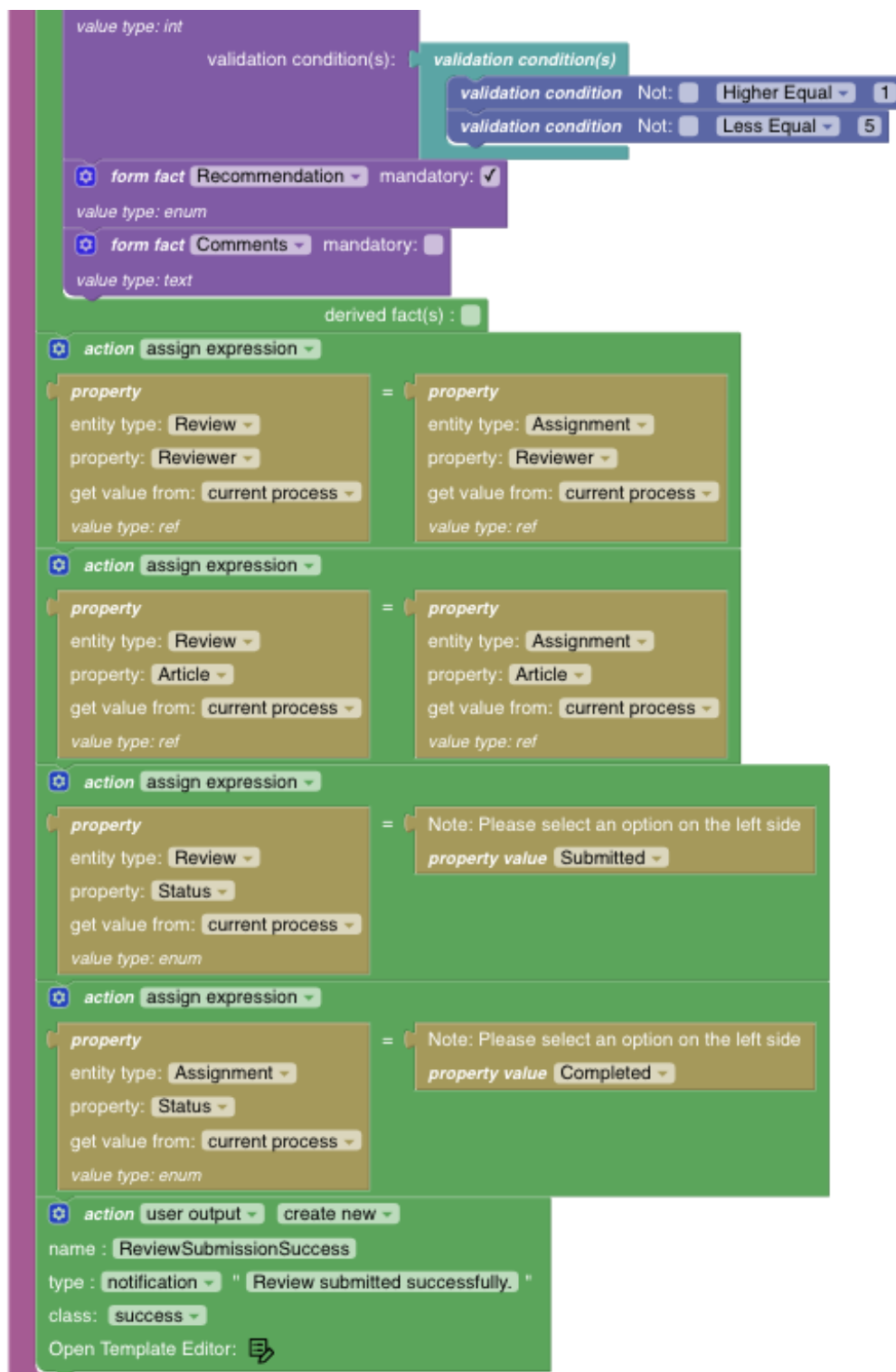


Figure 5.4: Continuation of the imported *Submit Review* draft: automatic assignments, status changes, and confirmation message.

Property type	Form component
Text	Text input
Numeric	Numeric input
Date	Date input
Time	Time input
Enumerated	Selection field
Entity reference	Selection field
Boolean	Radio buttons
File	File input

Table 5.2: Form components generated for each property type.

options. However, the importer does not reproduce every condition of an Action Rule as a validation condition in the Form. Those rules remain part of the active Action Rule and are handled separately by DISME.

After adding a submit button, the importer sends the Form.io JSON and the associations between its fields and the corresponding Action Properties to the DISME API. The platform stores the Form, its content, and these associations. Each Form is created through a separate request, so a failure does not remove Forms that were created earlier. If the step is repeated, the importer keeps the existing Forms and attempts to create only those that are still missing.

The end result is an initial version of the Form with the fields and organisation required by the generated rule. It uses a standard generated layout, which users can later modify through the Forms Manager in DISME. Currently, the importer generates one standard Form for each eligible Action and does not generate GenderMag variants.

5.6.5 Execution in DISME

Although importing the reviewed models and activating their Action Rules allowed some workflows to run through the existing DISME components, some rules still needed additional data at execution time. This included information about the domain entity associated with the logged-in user and the entity from which a transaction was initiated. Therefore, the runtime and UserApp were expanded to supply this data to the System Executor.

These changes allow the imported models, queries, Action Rules, and Forms to run through the existing DISME components. The wizard does not create the UserApp used to view records and start transactions in DISME. Consequently, users must be assigned to the appropriate roles and, when the current user is referenced, their accounts must also be associated with the corresponding domain entities. Both of these administrative configurations must be performed in DISME.

5.6.5.1 Current user and runtime values

An Action Rule may use the current user when assigning a reference or providing a parameter to a query. The existing `current_user` term returned the identifier of the authenticated account, while a property of type `prop_ref` expects the identifier of a domain entity, such as an Author or Reviewer. At runtime, the entity related to the account is retrieved, and its type is checked to determine whether it corresponds to the type expected by the reference. This resolution mechanism is used in the same way for query parameters referring to the current user. Values based on the current date or time are also resolved when the rule or query is executed instead of being fixed during generation.

5.6.5.2 Starting transactions from an existing entity

Another issue occurred when a transaction had to start from an existing entity. In the CONFERENCE system, for example, a Reviewer starts the submission of a Review from a particular Assignment. UserApp list components were extended with an `action_transaction_type_id`, allowing each row to start the configured transaction and send the selected entity as `origin_entity_id`. Before starting the transaction, the back-end checks that the component is associated with that transaction and that the selected entity belongs to the query results available to the authenticated user. Next, after these conditions have been verified, the origin is processed through the execution context and can be used for reading properties, resolving references, or editing the selected entity. This prevents access to an entity without sufficient context when there are multiple possible instances.

5.6.5.3 Rollback and form submission

The runtime also required changes to support `rollback_transaction`. Although the block already existed in DISME, its complete path from Blockly to execution was not available. Support was added to the code generator, persistence layer, and Execution Engine.

Another problem was that the Form is submitted in one request and the Action Rule continues in another. The data submitted through the Form may therefore have been stored before the rollback is reached. A compensation mechanism was added to remove data created by that submission, but only when it belongs to the same transaction and can be removed without affecting other data. We also removed the automatic retry logic from requests that update the execution state, and we no longer show a success message until the remaining actions of the rule have been completed.

5.7 System Creation Example

This section brings together the previous implementation steps through the CONFERENCE case. Rather than repeating the internal operation of each phase, it follows the

artefacts required to submit a review, from the initial business description to their execution in DISME. A detailed walkthrough of the wizard interface and each action required from the user is provided in Appendix D.

The example began with a conversation in Portuguese. In the first answer, translated here to English, the user provided a short description of the organisation and its users:

We organise scientific conferences. The system will be used by authors, reviewers, and the conference chair to manage submissions, reviews, and decisions in one place.

The following answers described the problems with controlling deadlines, following submissions and reviews, and limiting access to information. The user also explained how articles are submitted and assigned, which information needs to be stored, and when reviews become visible.

From this conversation, Phase 0 produced an initial draft containing *Article*, *Author*, *Reviewer*, *Assignment*, *Review*, and *Decision*. During the review, the user removed *Decision* as a separate entity and added *Configuration*, *Authorship*, and *Conflict Declaration*. The entity list confirmed for Phase 1 therefore contained 8 entities. This review took place before their properties and relationships were defined.

Table 5.3 summarises how the initial description was developed through the wizard and later used in DISME. The table presents the complete CONFERENCE model at a general level while identifying the artefacts used by the *Submit Review* example.

Stage	Input	User action	Result
Phase 0	Initial business description	Answer the interview and review the entity list	Business draft with 8 confirmed entities
Phase 1	Confirmed entity list	Describe and review each entity	State Model with 8 entity types and thirty-three properties
Phase 2	Confirmed State Model	Describe the roles, processes, and transactions	Process Model with 3 roles, 5 processes, and 13 transactions
Phases 3 and 3b	Imported State and Process Models	Describe and review the required queries	Query definitions using the identifiers returned by DISME
Phase 4	Confirmed models, queries, and transaction interviews	Review and approve the generated rules	Action Rule IR, rule representation, and Blockly XML
DISME integration	Reviewed models, queries, and rules	Open and save the imported draft and request Form creation	An active Action Rule and its initial Form
Execution	Configured users, roles, domain entities	Select a pending assignment and submit its review	A new review and an updated assignment

Table 5.3: Creation and execution of the CONFERENCE example.

For the part of the system followed in this example, the State Model defines both *Review* and *Assignment*. A review stores its score, recommendation, comments, and state,

together with references to the corresponding article and reviewer. An assignment records the article, reviewer, relevant dates, and its current state. These references allow the rule to create a review for the article and reviewer associated with a particular assignment.

The Process Model identifies *Reviewer* as the role responsible for *Submit Review*. This transaction belongs to the review process and creates a new instance of *Review*. Once the State and Process Models had been reviewed, the importer created or reused their records in DISME and stored the identifiers returned by the platform. The queries and Action Rules created later could then refer to the imported entities, properties, transactions, and permitted values.

Phase 4 collected the behaviour of *Submit Review*. The final reviewed rule presents the fields *Score*, *Recommendation*, and *Comments*, validates the score, associates the review with the article and reviewer from the selected assignment, sets the review state to *Submitted*, and changes the assignment state to *Completed*. Section 5.5.6 describes this rule and its representation in more detail.

After the rule had been validated, the Blockly Bridge converted its IR into the rule representation and Blockly XML. After user approval, the importer sent the XML to the Action Rules Manager as a draft, which the user then opened and saved as an active rule. Figures 5.2 and 5.3 show these two review points. Once the active rule and its Action Properties were available, the importer created the initial Form used to collect the 3 review fields.

The transaction also needs to know which assignment is being reviewed. Since the wizard does not create the UserApp, a Reviewer application was configured in DISME with a list based on the query `PendingAssignmentsForCurrentReviewer`. The user account was assigned to the *Reviewer* role and associated with the corresponding domain entity. Each row of the list can start *Submit Review* and pass the selected assignment as `origin_entity_id`. This configuration ensures that the rule uses the assignment selected by the reviewer instead of choosing one without enough context.

Figure 5.5 shows the Form presented when *Submit Review* is started in DISME.

When the Form is submitted, the rule creates the review, associates it with the article and reviewer from the selected assignment, and stores the values provided by the user. It also changes the assignment state to *Completed*, which removes it from the list of pending assignments. The example therefore follows one path from the initial description to its execution, although the UserApp and user associations still require configuration in DISME. Chapter 6 evaluates the wizard through a separate reduced conference scenario, while Appendix D provides the complete sequence of user actions.

5.8 Evolution from the Initial Proposal

The initial proposal was prepared before the complete wizard and its integration with DISME had been implemented. The overall strategy was maintained: the user describes a system in natural language, an LLM helps convert that description into DISME artefacts,

Fill the following form

Score *

Recommendation *

Comments

Submit

Figure 5.5: Execution Form for *Submit Review* in DISME.

and the generated results are reviewed before they are used by the platform. The design changed as the dependencies of the DISME API, Action Rules Manager, and System Executor were tested.

Table 5.4 summarises the changes from the initial proposal and the integration requirements identified during development.

Table 5.4: Changes made during implementation.

Aspect	Change
Context	The proposal included a Retrieval-Augmented Generation component to retrieve the organisation's existing ontology. The wizard instead builds context from the current session, confirmed models, metadata formats in the prompts, and imported identifiers. It has no separate retrieval system or external document ranking.
Review and activation	The proposed <code>validation_status</code> of <code>TO_VALIDATE</code> , with review through changes to the System Specifier and Action Rules Manager, was not implemented. State and Process Models are reviewed and confirmed in the wizard before import, since the structural endpoints create active records. Action Rules use the existing draft mechanism; opening and saving a draft creates or replaces the active rule.
Rule conversion	Direct conversion from generated JSON to relational records was replaced by the Blockly Bridge. It builds blocks in a headless workspace and exports XML, reusing DISME's block definitions and code generators instead of implementing the same conversion in Python.

Aspect	Change
Corrections	Users review and correct artefacts in the wizard before import. Action Rules are shown as generated actions with natural-language explanations. When the user requests a change, the wizard regenerates the rule using that feedback. The explanations remain in the wizard and are not added to the Action Rules Manager.
Queries and forms	Execution tests showed that queries must be imported before the Forms and Action Rules that use them. This led to separate form queries in Phase 3 and Action Rule queries in Phase 3b. Forms are created in a separate step after drafts are opened and saved, when the required Actions and Action Properties become available.
Execution	The CONFERENCE case revealed missing runtime data. Changes to the runtime and UserApp provide the current user's domain entity and the entity from which a transaction starts. Support was also added for runtime date/time values and complete <code>rollback_transaction</code> execution (Section 5.6.5).

These changes support the selected workflows. The wizard still does not create the UserApp or configure the associations between users, roles, and domain entities; these tasks remain in DISME.

VALIDATION

6.1 Context

Chapter 5 described the implemented wizard and its integration with DISME. The validation considers the complete path from a description of a business system to the review, import, and execution of the resulting artefacts. Overall, the goal is to determine whether the workflow can produce artefacts that can be imported into DISME and if participants with no prior experience of the wizard can follow and complete each step on the wizard pipeline, after a short familiarisation. The specific goals and guiding questions are identified in section 6.2.1.

Validation is needed because a syntactically valid output is not necessarily a correct system specification. The output of an LLM may vary between runs, omit information supplied by the user, or represent a requirement incorrectly. In addition, artefacts that satisfy their respective schemas can reference elements which do not currently exist in the model. Artefacts that are valid in isolation could also fail when they are imported together or interpreted by the runtime.

The implemented validators are focused on the constraints which can be verified by comparing with a well-defined source of truth, such as required fields, supported structures, identifiers, and references between artefacts. This means that these checks cannot tell if a valid structure accurately represents the intended business process. Therefore the user has to have a way to evaluate the resultant artefact, and to accept or correct it before using it. In addition to validating the artefact itself, human review is also an integral component of the workflow under evaluation.

This validation is based on the evaluation activity from the Design Science Research methodology discussed in Section 1.4. The format for this validation's reporting has been developed according to the guidelines for reporting controlled experimentation in Software Engineering provided by Jedlitschka et al. [53]. This chapter adopts the elements applicable to the present study, including goals, participants, materials, tasks, procedure, analysis, discussion, and threats to validity. Experiment-specific elements, such as treatments, group allocation, causal hypotheses, and hypothesis testing, are not

used.

The accompanied sessions use the same version of the wizard, the same conference scenario, and the same base materials and procedure. All participants use the wizard; there is no comparison group or alternative method. The study is therefore not a controlled experiment. It describes what happens under the defined conditions using data recorded by the system, observations made during the sessions, and questionnaire responses. The analysis is mainly descriptive.

The participant study targets users who have not used the wizard before and may have different levels of familiarity with DISME. Prior knowledge of the conference domain is not required because the domain and the common business facts are introduced through the study materials. The evaluation concerns their use of the selected workflow rather than their ability to recall the scenario or define every feature of a conference management system.

The study does not compare the wizard for DISME with manual modelling. Lastly, using a single scenario and a limited participant sample limits our ability to apply the findings to other domains, systems, or users.

6.2 Study Planning

This section defines how the accompanied study is planned before data collection, including its goals, participants, conditions, materials, tasks, and measures.

6.2.1 Goals

The goal of the validation is to see whether users with no previous experience with the wizard can use the LLM assisted workflow to create and review an initial specification of a simple conference management system, and whether the confirmed artefacts can be imported and used in DISME. As well as the final result, the validation examines the quality of the generated artefacts, the changes made during review, the assistance required to complete the workflow, and the participants' perception of the wizard.

Table 6.1 summarises the questions that guide the validation and the main evidence used to answer each one. These are study questions used to connect the goals with the collected evidence. They are not additional research questions for the dissertation and are not causal hypotheses.

Table 6.1: Guiding questions and main sources of evidence.

Guiding question	Main evidence
Can the defined workflow produce artefacts that can be imported into DISME and used to execute the reduced conference scenario?	Pre-study technical verification, participant session artefacts, import results, and runtime execution.
Can participants complete the tasks in the common workflow?	Task completion, time, assistance, and observation notes.
Do the artefacts correctly represent the facts in the scenario?	Quality assessment of the first and confirmed results against the reference specification.
Can participants review the generated artefacts and, when necessary, identify and correct differences from the defined scenario?	Requested corrections, changes between the first and confirmed results, observations, and questionnaire responses.
How do participants assess the clarity and usability of the wizard, their understanding of what is imported, their confidence in the confirmed result, its perceived usefulness, and the workload perceived during the common workflow?	System Usability Scale, NASA-TLX, study-specific statements, and open responses.

The free exploration at the end of each session has a secondary goal. It observes what participants decide to do when they are no longer following the common workflow, which topics they choose to discuss with the wizard, and what difficulties they encounter. Since this does not get a semantic quality rating, the outcome of the free part is used as an independent analysis and does not affect the success determination of the common workflow.

6.2.2 Participants

Twenty participants took part in the formal sessions. As explained in Section 6.4.1, 1 session was excluded during data preparation, leaving 19 sessions in the analysed sample. The target users are people who can describe a business process in natural language but have not used the wizard before. Previous knowledge of DISME, DEMO, Blockly, or formal modelling is not required. Experience with the conference domain is also not required because the common business facts are available in the participant script, which the user has access to throughout the test.

Recruitment followed a convenience method by using computer science students and researchers who are accessible to the researcher and had never used the wizard before this study. Participation is voluntary, and no compensation is provided. In addition to being available, participation requires the individual to understand all study materials, to communicate in Portuguese, and to participate remotely via TeamViewer. Familiarity

with DISME, low-code platforms, generative AI, or conference processes was not used as a selection or exclusion criterion.

The study information is provided in advance, and informed consent is obtained before any study data are collected. This step takes place before the 65-minute session and is not included in its duration. Participants may withdraw without providing a reason.

The planned sample size is not based on a statistical power calculation because the study does not test causal hypotheses or compare experimental groups. The number of participants in this study was determined by an estimate for the number of sessions needed to provide evidence that would allow observation of the same workflow across multiple cases. This target was considered sufficient to report the observed actions and outcomes using counts and descriptive statistics.

Previous experience is recorded using broad categories rather than personal or professional details. The questionnaire asks about familiarity with conference processes, experience with low-code or modelling tools, frequency of generative AI use, and previous experience with DISME. These characteristics are used to describe the final sample and help interpret individual cases. Names, email addresses, gender, age, and job titles are not required for the planned analysis and are not collected in the study questionnaire.

Formal session records are identified only by codes from P01 to P20. Only the researcher and the supervisors have access to the collected data. No audio, video, or screen recordings are made.

6.2.3 Study Design

The sessions use the wizard's study mode. Normal mode, described in Chapter 5, allows more open-ended exploration of the proposed system, including additional questions to clarify requirements and examine business processes in greater detail. This may prolong the interview and extend its scope beyond the tasks selected for evaluation. Study mode therefore guides the conversation through predefined topics to keep the sessions focused on the evaluated workflow and manageable within the planned duration.

These topics cover the system's purpose and users; the information to be stored and its constraints; roles and responsibilities; process sequence and completion; and the form inputs, automatic updates, preconditions, subsequent actions, and confirmations associated with each transaction.

The study does not compare groups or alternative methods and has only one study condition. When analysing the results generated for each artefact item during a single session, they are viewed as measures from the same unit rather than as independent observations.

Each session is conducted individually and using TeamViewer to provide a remote connection to the researcher's computer.

The study conditions are kept the same across the formal sessions. They include the wizard version, prompts, language models, model parameters, initial database snapshot,

test account, participant materials, example file, time limits, and moderation rules. All these parameters are tested in a pilot, then "frozen" before formal data collection began.

The business facts and task order are fixed, but the exact conversation is not. The language model may ask questions in a different order, request clarification, or formulate the same question differently. The participant script provides the business facts needed for the session, rather than a fixed list of answers. Participants formulate their responses using these facts, since the interviewer's exact questions may vary. Participants answer in their own words while consulting and adapting the information in the script, and can ask the wizard to explain or reformulate a question. Conversations and generated artefacts are preserved so that this variation can be considered during the analysis.

Table 6.2 separates the materials available to the participant from those reserved for the researcher. This prevents the reference answer and the quality criteria from directing the participant's decisions.

Table 6.2: Materials used in the participant study.

Access	Materials
Participant	Study information and consent; the VetClinic familiarisation environment; the conference scenario script; the wizard; the NASA-TLX HTML; and the final questionnaire.
Researcher only	Familiarisation guide; moderator guide; observation form; expected result and quality assessment criteria; session run-book; test PDF; and questionnaire analysis guide.

The test PDF is the same short document in every session and contains no personal or real research information. It is used only to fill the required file field when the moderator submits the example article in DISME. Its content is not part of the quality assessment.

Each session is planned to last 65 minutes, distributed as shown in Table 6.3. The information sheet is sent and informed consent is obtained before the session; neither is included in the 65 minutes.

Table 6.3: Planned distribution of a participant session.

Part	Planned duration
Practical familiarisation	10 minutes
Common workflow in the wizard and DISME	35 minutes
NASA-TLX and response recording	3 minutes
Free exploration	8 minutes
Final questionnaire and comments	7 minutes
Transitions and margin	2 minutes
Total	65 minutes

The common workflow clock starts when the moderator selects **Start common workflow**, and includes reading the script. The scenario represents a reduced submission

and decision process. It contains one main entity, *Article*, with the required properties *Title*, *Abstract*, *File*, and *Status*, two roles, *Author* and *Conference Chair*, and two transactions, *Submit Article* and *Record Decision*. No additional entity, relationship, or query is required by the scenario.

The author provides the article data and submits it. The submission creates the article, sets the current state of the article to *Submitted*, makes the decision available and displays a confirmation message. The Conference Chair can then select *Accepted* or *Rejected* for an article whose current state is *Submitted*. The decision updates the current state, presents a confirmation, and ends the process. During the common execution checks, the moderator selects *Accepted* so that the expected final state is the same in every session.

This scope was developed with the intention that all aspects of the process in this session, from description through to actual run time, can be viewed by the researcher. As a result, participants do not have to create the full CONFERENCE system, or to access internal JSON, modify Blockly blocks, or to build the system directly via the DISME interface.

Distribution of responsibility during the common workflow is summarised in Table 6.4.

Table 6.4: Responsibilities during the common workflow.

Part	Participant	Moderator
Interviews	Describes the system and answers the wizard's questions.	Observes and records relevant events.
Review	Inspects, corrects when necessary, and confirms each result.	May explain the interface, any semantic or technical assistance is recorded separately.
Import	Starts or confirms the import operations presented by the wizard.	Intervenes only if a technical failure prevents continuation.
DISME setup	Observes the required configuration.	Assigns the test account to the imported roles, activates the rule drafts, and requests form creation.
Execution	Observes and may comment.	Submits the example article, records the decision, checks the final state and history, and records outcomes and incidents.

6.2.4 Measures

The measures are defined before the formal sessions and linked to the goals in Section 6.2.1. This follows the recommendation to state what is measured, how it is measured,

and how each measure supports the study goals [53].

The study uses the concepts of effectiveness, efficiency, and satisfaction from ISO 9241-11 as a way to organise the usability evidence [54]. The effectiveness is evaluated by whether or not participants are able to accomplish tasks, and whether they produce correct results. Efficiency is also assessed using time, number of interactions, and amount of required assistance. Satisfaction is assessed with the participant’s questionnaire.

Table 6.5: Measures collected during the study.

Dimension	Operational measure and scale	Main source
Task completion	Each task is recorded as completed or not completed. Assistance is recorded separately as none, procedural, semantic, or technical.	Observation form and session record
Time and interaction	Duration in seconds; counts of questions, reformulations, corrections, regenerations, phase revisits, warnings, and import errors.	Automatic session record and observation form
Artefact quality	30 criteria scored with 1, 0.5, or 0. NC indicates that the result could not be scored.	Initial and confirmed artefacts, expected result and quality assessment criteria
Functional result	Import, configuration, and execution checks recorded as passed, failed, or not checkable (NC).	Wizard, DISME, and execution record
Participant perception	10 SUS items and 5 study-specific statements about question clarity, result review, ability to identify and correct incorrect information, understanding of imports, and perceived usefulness, followed by open questions.	Final questionnaire
Perceived workload	6 ratings: mental demand, physical demand, temporal demand, performance, effort, and frustration. Values are stored in the orientation displayed by the HTML questionnaire.	NASA-TLX HTML and observation form
Free exploration	Conversation topics, questions, assistance, difficulties, comments, and technical incidents. No reference score is assigned.	Separate automatic record and observation notes

The common workflow contains seven assessed participant tasks. Their completion conditions are defined in Table 6.7, and their outcomes are recorded in the observation form.

Task completion and assistance are kept separate. Therefore, a task can be completed and still require support. Task completion is either *completed* (the desired, observable outcome has been achieved prior to the conclusion of the common workflow, regardless of the assistance) or *not completed* (the task was not achieved). Assistance uses the following

categories:

- **Procedural assistance (P):** help about how to operate the interface, such as indicating where an available control is located, without giving the expected business answer;
- **Semantic assistance (S):** help that explains part of the conference scenario or indicates content that should be included in an answer or artefact; and
- **Technical assistance (T):** an unexpected intervention needed to recover from a failure so the participant task can continue.

More than one category may be recorded for the same task. The moderator actions planned in Table 6.4 are not assistance because every session requires them and the participant is not expected to perform them. While completing a task with procedural assistance counts as a successful completion, semantic or technical assistance prevents that session from achieving the strict success criteria defined below.

These assistance categories are specific to this study rather than a standardised scale. The common instructions provided to every participant are also not counted as assistance. Recording the categories separately therefore avoids treating these situations as equivalent.

The common workflow duration is measured from the moderator selecting **Start common workflow** until the seven participant tasks and the moderator’s DISME checks are completed, or until a technical failure prevents safe continuation. Reaching 35 minutes is recorded as exceeding the planned duration, but does not by itself stop the workflow. The total duration and the duration of planned moderator activities are stored separately. Technical assistance and incidents are recorded, but a separate recovery duration is reported only when a distinct start and end time can be identified. Counts of the number of questions, reformulations, corrections, regenerations, revisits, warnings and import errors document the way the process has been accomplished. They are not treated individually as success or failure: for example, a correction can indicate either an initial generation problem or effective use of the review mechanism.

Artefact quality is assessed twice: for the first complete result presented for review and for the result confirmed by the participant. Both are compared with the same reference specification for the reduced conference scenario. The rubric contains 30 criteria: 2 for the initial overview, 7 for the State Model, 8 for the Process Model, 2 that check that no unnecessary queries are introduced, and 11 for Action Rules. These criteria were derived from all the necessary facts and behaviours that the participant script requires.

The initial result is the first complete result presented for review. It includes any automatic processing, validation, and refinement performed by the pipeline before that review. The initial quality score therefore characterises the result presented by the pipeline, rather than the unprocessed output of an isolated LLM call. The confirmed score characterises the result after the participant’s review and any subsequent changes.

The complete set of criteria and their associated critical failures is shown in Appendix B, Table B.1. Each applicable criterion is scored as follows:

- 1: the required fact or behaviour is correct and complete;
- 0.5: the criterion is partly satisfied, following the scoring rules for that criterion;
- 0: it is absent, incorrect, contradictory, or invalid; and
- NC: the relevant artefact was not produced or the available evidence is insufficient to score it.

All 30 criteria have the same weight:

$$Quality\ score = \frac{\sum_{i=1}^{30} c_i}{30} \times 100, \quad c_i \in \{0, 0.5, 1\}. \quad (6.1)$$

A set containing NC does not receive a percentage and is reported as incomplete. Any equivalent names entered by participants are scored with full credit if the name preserves the necessary concepts of the referenced business, responsibilities, references, and run-time effect.

Some failures are marked as critical because a high average quality score could otherwise hide a problem that prevents the intended workflow. This type of failure includes missing a required entity or transaction, an invalid structural or referential dependency, an Action Rule performing the incorrect operation, failed required imports, a false indication of success, participant data from multiple session entries being intermixed, or failure to reach the required final state during execution. Critical failures are recorded separately from the numerical quality score. A critical error means that an important requirement of the scenario is not met, even if the tested workflow can still be completed.

The researcher applies the rubric to every result. Deterministic validators and the recorded import and execution outcomes are used for structural and functional criteria. Semantic criteria are judged from the saved artefact and the scenario, with the reason for any partial or zero score recorded beside the criterion. Since there is a single evaluator, inter-rater agreement cannot be calculated.

The difference between confirmed and initial quality is calculated only when both versions have a complete score:

$$\Delta Q = Q_{\text{confirmed}} - Q_{\text{initial}} \quad (6.2)$$

This value shows whether the review preserved, improved, or reduced correspondence with the scenario. The correction rate is calculated only for sessions whose initial result contained at least one correctable difference. Improvements should not be required if the reviewer did not need to make changes as all requirements were met in the first result.

The functional result is documented separately from the quality score. The check records whether structural artefacts and Action Rules are imported into the intended system, whether rule drafts can be inspected and activated, whether all required forms may be generated, whether both the submission and decision transactions can be executed

for the same article, and whether its current Status is Accepted while Submitted remains in its history. Separate checks prevent a semantically reasonable explanation from being considered evidence that the process was successfully completed.

Perceived usability is measured with the 10 item System Usability Scale (SUS) [55], using a published European Portuguese version [56]. Each item is answered from 1, corresponding to strong disagreement, to 5, corresponding to strong agreement. For the positively worded odd items, 1 is subtracted from the response. For the negatively worded even items, the response is subtracted from 5. The 10 adjusted values are added and multiplied by 2.5, producing a score from 0 to 100. This value is a SUS score and is not interpreted as a percentage. A SUS score is not calculated when an item is missing. In the questionnaire instructions, the product evaluated by the SUS is the wizard as experienced during the practical familiarisation, common workflow, and free exploration. The technical steps performed by the moderator in DISME and the quality of the TeamViewer connection are excluded from this assessment.

Perceived workload during the common workflow is assessed using the six NASA Task Load Index (NASA-TLX) dimensions: mental demand, physical demand, temporal demand, performance, effort, and frustration [57]. Participants complete the HTML questionnaire after the common workflow checkpoint is saved and before free exploration. The assessment concerns their own use of the wizard, excluding familiarisation, free exploration, and the moderator's technical operations in DISME. The six ratings produced by the HTML questionnaire are saved in the observation form. In this HTML, the performance item is evaluated from lower to higher perceived success, so higher values indicate better performance.

The questionnaire also contains five statements specific to this study: clarity of the wizard's questions, support provided by the review screens, ability to review and correct the generated content, understanding of what would be imported, and perceived usefulness for producing an initial specification. Each statement uses the same 5-point response scale as the SUS items, but the statements are analysed separately and are not included in the SUS score. There are open questions requesting users to identify the most difficult part, moments of doubt, and suggestions for future improvement that could make the wizard clearer or more useful. The exact wording and response scales of the NASA-TLX questionnaire and final questionnaire are provided in Appendix C.

Table 6.6 defines the criteria selected before data collection. They are decision rules for this study and its fixed scenario, not universal usability thresholds or estimates of comparative improvement.

Table 6.6: Planned criteria for interpreting the study results.

Criterion	Planned interpretation
Strict session criterion	The common workflow is completed within 35 minutes; all assigned tasks are completed; assistance is at most procedural; confirmed artefact quality is at least 85%; there is no critical failure; both transactions run for the same article; and its current Status is Accepted while Submitted remains in its history.
Overall feasibility	At least 16 sessions satisfy strict session success, and no participant data are mixed between sessions.
Review does not worsen the result	In at least 18 sessions, confirmed quality is equal to or greater than initial quality and review introduces no critical failure. A session without both scores does not satisfy this criterion and is reported separately.
Correction of initial differences	Among comparable sessions containing at least one correctable initial difference, at least 70% finish with a higher quality score. The numerator and denominator are always reported, and this result is treated as supporting descriptive evidence rather than as a study-wide success criterion.
Perceived usability	The mean of the complete SUS scores is at least 70, with the number of valid scores also reported. This planned reference is close to the acceptability boundary reported in prior work on interpreting SUS scores [58].
Study-specific perceptions	For each of the five statements, the median is at least 4 and at least 14 responses are 4 or 5. These results support interpretation but do not replace the SUS score or observed evidence.
Security and record integrity	There are no silent invalid imports, false success indications, or cases in which data from different participants are mixed.

There is no established threshold for the study-specific rubric. The 85% quality threshold is therefore a planned decision rule and corresponds to at least 25.5 of the 30 rubric points. It permits small omissions while requiring most scenario facts and behaviours to be represented, and the critical-failure rule prevents a high total from compensating for a failure that blocks or changes the process. The targets of 16 strict sessions and 18 non-worsening reviews were defined from the planned sample of 20, corresponding to 80% and 90%, respectively. After the exclusion of P17, these numerical targets were retained rather than recalculated from the analysed sample. The correction and study-specific perception criteria provide supporting evidence rather than claims about a wider population. No single combined score is created from task completion, quality, execution, and participant perception.

6.3 Execution

This section documents how the study was carried out under the conditions defined in Section 6.2. It separates the preparatory checks from the formal participant sessions. For each stage, the execution record includes the completed and incomplete cases, missing data, technical incidents, and any deviation from the planned procedure.

6.3.1 Preparation and Pilot

Before participants were involved, a controlled technical rehearsal was conducted on 10 September 2026, using the reduced conference scenario defined in Section 6.2.3. The purpose of this technical rehearsal was to verify that the complete path could work in a clean environment and also to detect technical problems before the participant sessions. This technical rehearsal was performed by the researcher and it did not count as a participant session or as a source of study data. Therefore, its artefacts, time, and corrections are excluded from the participant study results.

The technical rehearsal used an isolated database named `disme_pilot_session`. It was restored from a verified baseline containing the platform configuration and no entity types, transaction types, Action Rule drafts, entity instances or transaction instances from previous executions.

However, the Status property was generated to accept multiple existing values. Consequently, this allowed the article to have both `Submitted` and `Accepted` as current values at the same time. Therefore, the Phase 1 interview was modified along with the deterministic normalisation to ensure that all lifecycle properties retain only one current value, while preserving previous values in history. For this case regression testing was applied to verify the correction.

The database was restored and the scenario was tested again with the wizard in a new session. This repeat run used the corrected interview and normalisation described above. The submission created an article with `Submitted` as its present state and made the decision transaction available. The decision then updated the present state to `Accepted`, while `Submitted` continued to be stored in the article's history. Both transactions reached their final DEMO state. After the completion of the rehearsal, there were 184 successful automated tests and 4 dependency warnings. These tests included unit, integration, and regression tests covering deterministic validation, import preparation, artefact conversion, interview and review safeguards, and previously identified faults.

The automated tests also check references between entities. A test model with 8 properties that refer to other entities is accepted when every target entity and property exists. If a target is missing, its property does not exist, or the reference uses the wrong entity type, validation fails before import.

The rehearsal also showed that the DISME dashboard did not refresh automatically after a transaction, although the following transaction had been created correctly in the

backend. Therefore, the moderator procedures include a refresh to the dashboard prior to determining if the transaction had been successfully completed.

Additional study controls were implemented after this rehearsal. These controls assign a unique code and folder location for each formal study session, they retain both the initial and confirmed artefacts, take a checkpoint at the end of the common workflow, perform free explorations on a duplicate, restrict importing from the duplicate, and produce a final proof package upon completion of each formal study session.

A pilot session was executed on 14 September, 2026 by 1 participant who had no experience using the wizard to test the instructions, order of tasks, timing, moderation and recording of data before the formal participant sessions. The pilot consisted of familiarisation with the wizard, the workflow with execution in DISME followed by NASA-TLX, free exploration of the system and a final questionnaire. The pilot is excluded from the formal sample.

Familiarisation took 8 minutes. The participant's activities in the wizard took 31 minutes, and the moderator's execution in DISME took 3 minutes, giving 34 minutes for the common workflow. NASA-TLX took 2 minutes, free exploration took 8 minutes, and the final questionnaire took 6 minutes. As such, the common workflow was completed within the target of 35 minutes.

The participant completed the seven wizard tasks, although semantic assistance was needed. The moderator explained how the participant's generated artefacts were going to get transferred to DISME, and he helped clarify 2 unclear interview questions that the participant had asked the wizard to reformulate. The main observed difficulties concerned understanding some questions and the generated model. Therefore, task completion was not achieved with full autonomy.

The moderator executed both transactions while the participant watched, and in both cases, the transaction was completed. At this point, the participant also confirmed that the current article status was Accepted. However, the decision selector displayed an unexpected numeric option, 1. The artefacts, observation records, six NASA-TLX responses, and final questionnaire responses were saved.

The procedure and planned time for the session remained unchanged after the pilot. The session provided evidence that the measured activities could be completed within the time established. It should be noted that this is only based on a single participant in a pilot study and does not show that all participants had the same experience.

6.3.2 Procedure

Each session follows the same procedure through TeamViewer, with the participant controlling the wizard on the researcher's computer. Before the timed activities, the moderator restores the approved database baseline, checks the services, test account, remote connection, and study session controls, and prepares the participant script. Informed consent is given by the participant before data collection and it is not part of the total session time.

The planned timing is shown in Table 6.3. If answering or saving the responses takes longer, the actual duration is recorded rather than rushing or stopping the participant to meet the planned time.

The session starts with hands on practice with the separate VetClinic example, called the familiarisation period. The moderator goes through the common guide explaining how to answer the wizard, how to recognise which phase the participant is currently in, what each result means, how to ask for corrections, and how to confirm a generated artefact. The participant tries these operations while receiving this guidance. No training artefacts are imported into DISME. At the end of the 10 minutes, the moderator selects `End` and `clear familiarisation` and checks that the training data are cleared before starting the formal session. These data are not included in the participant results.

The moderator creates a formal session with the participant's anonymous code and makes the conference script available. At this time the participant is informed that they may consult the conference script during the common workflow as much as necessary. The participant also learns that they do not have to use word-for-word what is written in the script. Once the participant is ready to proceed, the moderator selects `Start common workflow`. Within this 35-minute target, the participant reads the script, responds to the interview questions, waits for the wizard to generate questions and artefacts, reviews the results, imports the confirmed artefacts, and this period also includes the moderator's checks in DISME. Environment preparation is not included.

The participant follows the seven tasks in Table 6.7. The order of the tasks is fixed, although the wording and order of questions within an interview may vary. Participants may consult, adapt, or copy information from the script and ask the wizard for an explanation or reformulation. After reviewing each result, they can apply corrections if that is needed. There is no need for correction when there is already a valid representation of the scenario. Also, no intentional errors are made in order to make the participant have to correct something.

Completion of the tasks, assistance, and artefact quality are all recorded independently. Confirming an incorrect model can complete a review task, but the difference is evaluated through the quality rubric. Simply clicking a confirmation button is not considered evidence that the user understands the generated results. An import task requires a recorded successful result, rather than only a click on the import button.

The moderator performs the required configuration and execution in DISME. After structural import, the test account is associated with the imported `Author` and `Conference Chair` roles. After rule import, the moderator loads each draft in the Action Rules Manager and uses `Save action rule` to activate it. The moderator then creates the forms through the operation available in the wizard, submits the example article, and records the decision `Accepted`. The checks establish whether both transactions finish for the same article, whether its current `Status` is `Accepted`, and whether `Submitted` remains in its history. The dashboard is refreshed before checking the available tasks and after each transaction. The participant may observe and comment, but these operations are not

Table 6.7: Participant tasks and conditions for completion.

Task	Condition for completion
1. Describe the system	The initial interview ends and the wizard presents the overview proposal.
2. Review the overview	The participant inspects the proposal and confirms the generated version.
3. Review the State Model	The participant inspects the entity and properties and confirms the model.
4. Review the Process Model	The participant inspects the roles, process, and transactions and confirms the model.
5. Import the structural models	The participant starts the import of the confirmed models and a successful result is recorded, with no unresolved import error.
6. Review the Action Rules	The participant inspects and confirms the rules for both transactions, making corrections if needed.
7. Import the rule drafts	The participant starts the import of the confirmed rules and a successful result is recorded, with no unresolved import error.

assessed as participant tasks or autonomous use of DISME.

After each participant task, the moderator records completion and any assistance provided, together with short notes on hesitations, difficulties, participant comments, and incidents. The start, end, and outcome of the moderator's technical operations are also recorded separately. Conversations, artefacts, and events recorded by the wizard are not transcribed manually. Counts and quality scores are checked afterwards using the saved files and notes.

If a participant becomes blocked, the moderator first allows an attempt to resolve the difficulty through the script or the wizard's own explanations. Additional support for a participant can be provided if necessary but it is documented along with both the support and impact of that help.

The common instructions, familiarisation, consultation of the script, use of the wizard's explanations, and planned moderator operations do not count as additional assistance. An error on the moderator's own DISME check is documented as an Incident within that block, however, this action alone does not count as a technical assistance to the participant.

The common workflow clock runs continuously through all of the different stages, such as model response, review, imports to DISME, any assistance necessary, planned moderator action, and technical recovery. It is not paused or restarted for these events, and the total session clock also continues. The common workflow ends when the seven participant tasks and the DISME checks are done, or when a technical failure does not allow the session to continue. If it finishes before the planned time, the session advances without having to fill the remaining time.

If the 35-minute target is reached before completion, the moderator records the overrun and allows the participant to continue. A workflow completed after this point is recorded as completed, but it does not satisfy the 35-minute condition of the strict session. If a technical failure prevents continuation, the available evidence is stored and the incomplete activities are reported with their reason.

When a technical failure occurs, the moderator records the time, phase, and a short description and informs the participant. A failed generation is retried once with the same answers. Before attempting another import, the moderator verifies if there was already some form of data imported so they do not duplicate this information. If the problems persist, a saved state can only be restored at a point in time where restoring does not conflict with imports already made. Code and prompts used for each participant are not modified. If a safe restore option is not available, the common workflow ends early with a recorded technical reason, and the remaining activities are carried out if possible. Any activity impacted by the failure is recorded as not performed, with its reason.

At the end of the common workflow, the moderator selects `Save common workflow checkpoint` and verifies that the artefacts are saved before continuing. NASA-TLX is then applied using the HTML provided by the supervisor¹, using its statements and scales as displayed. In the performance dimension, a higher value represents greater success. The participant is told to evaluate their own use of the wizard throughout the common workflow, but not including the familiarisation phase, free exploration, and the technical work performed by the moderator in DISME.

After all six scales have a response, the moderator uses `Copy results`, pastes the copied values into the observation form under the participant code, and verifies that they are saved. Their order is preserved: mental demand, physical demand, temporal demand, performance, effort, and frustration.

Free exploration begins only after the NASA-TLX responses have been saved. The moderator selects `Start free exploration`, which opens a separate conversation based on a copy of the common workflow checkpoint. The participant can choose the subject of the new conversation and use it to understand the generated system better, propose extensions, or discuss systems from other domains. Model updates or importing new information do not happen in this mode. Its records are kept separate and do not change the quality score or completion status of the common workflow. After 8 minutes, the moderator selects `End free exploration`. The chosen subject and relevant difficulties or comments are noted without manually copying the conversation.

Finally, the participant answers the Google Forms questionnaire, including SUS, the study-specific statements, and open questions, using the same anonymous code. The moderator selects `Close session` and checks three sources separately: the wizard's saved session package and checklist, the observation form including the NASA-TLX values, and submission of the final questionnaire under the correct code. The wizard's checklist

¹<https://eelab.arditi.pt/nasa-tlx/nasa-tlx.html>

does not by itself confirm the other two sources. Missing or incomplete records are documented and preserved, and the database is not restored for another session until the existing session files have been checked and saved.

6.4 Analysis

The analysis combines the automatic session records, the moderator's observation form, the initial and confirmed artefacts, the execution evidence from DISME, the NASA-TLX responses, and the final questionnaire responses. The results are then presented in the same order as the study goals, while keeping observed behaviour, technical outcomes, and participant perception separate.

6.4.1 Data Preparation

The participant session is the main unit of analysis. The pilot is not included in the formal data set. Twenty formal sessions were conducted. During data preparation, P17 was excluded because the moderator had stopped the common workflow at the 35-minute target instead of allowing the participant to complete the procedure and recording the overrun. Since the missing outcomes resulted from this procedural deviation, the analysed data set contains 19 sessions. The original P17 records were retained but were not used in the analyses. Records from the remaining sources are joined only through the anonymous code assigned to each participant. Before calculating a measure, the session package is checked for the correct participant code, version information, autosave, first and confirmed artefacts for the completed phases, checkpoints, final evidence, and separation of the common workflow from free exploration. In cases where any of these checks fail, the failure is documented and reported but the original data are not modified. The NASA-TLX record is checked for the participant code, the presence of all six ratings, and their order: mental demand, physical demand, temporal demand, performance, effort, and frustration.

The familiarisation data are excluded from the evaluated workflow. Data produced during free exploration are also separated from the common workflow. A session is not removed merely because a task was incomplete, assistance was required, or a technical incident occurred. Instead, the available evidence is retained and the reason for the incomplete result is recorded. Each calculation states its denominator. As such, missing values are never estimated. Artefact sets containing NC do not receive a quality rating. Questionnaire responses with one or more missing items cannot have a SUS score, while if one or more NASA-TLX ratings are missing, the missing values are not estimated.

6.4.2 Analysis Procedure

The sample is described by participants' previous familiarity with conference processes, low-code or modelling tools, generative AI and DISME. However, they are not used to

create comparative groups. Any patterns that appear are reported only as exploratory observations and not as statistically supported conclusions.

Task completion is reported for each task as a count and percentage of the available sessions. Autonomous completion means that the task was completed without procedural, semantic, or technical assistance. It is possible for a single session to receive multiple types of assistance and therefore, these categories are not mutually exclusive and their counts are not expected to sum to the number of sessions. The analysis also reports how many sessions meet every condition of the strict success criterion in Table 6.6.

For the common workflow and its main parts, time is summarised using the mean, standard deviation, median, and range. Remaining workflow time and the time of planned moderator operations are reported separately. Technical recovery time is reported separately only when a distinct interval was recorded. Otherwise, it remains included in the remaining workflow time. The number of wizard questions, participant requests for clarification or reformulation, and requested corrections are reported as totals and as the number of sessions in which they occurred. A correction, for example, is not classified by itself as either a problem or a successful outcome.

The quality rubric is applied to the first result presented for review and to the result confirmed by the participant. The 30 rubric criteria are scored separately but are not treated as independent observations; the participant session remains the unit of analysis. For each session with two complete scores, the paired difference defined in Section 6.2.4 is calculated. In addition, the analysis shows the number of sessions in which there was an increase, no change, or a decrease in the quality. It also provides information about the distributions of the initial scores, confirmed scores, and differences for all comparable sessions.

The correction rate is calculated as the number of comparable sessions with $\Delta Q > 0$ divided by the number of comparable sessions containing at least one correctable initial difference. The numerator and denominator are reported explicitly.

Functional outcomes are analysed separately from the quality rubric, using the session checks defined in Section 6.2.4. Failures are grouped by their recorded origin: wizard, model, DISME, configuration, network, test data, or study material. Recovered failures remain within the collected data and are associated with the appropriate technical assistance.

The responses to the 10 SUS items produce one score per participant using the calculation defined in Section 6.2.4. The distribution is described through the mean, standard deviation, median, minimum, and maximum. The sample mean is compared with the planned reference value of 70. All complete questionnaires are included in this calculation, even when the participant did not complete the common workflow, because the questionnaire records the experience that occurred rather than only successful sessions.

The six NASA-TLX dimensions are analysed separately. For each dimension, the number of valid responses, mean, standard deviation, median, minimum, and maximum are reported. Higher ratings indicate greater perceived workload, except for performance,

where a higher rating indicates better perceived performance, according to the HTML questionnaire. No overall score or pass threshold is calculated, and the ratings are not combined with SUS or the other study measures.

The five study-specific statements are analysed separately. For each statement, the results present the distribution from 1 to 5, the median, and the number and percentage of responses equal to 4 or 5. The statements are not added together and are not combined with SUS. The response about TeamViewer is used only to interpret remote-access incidents and does not enter the wizard’s usability, quality, or success measures.

The open questionnaire answers and observation notes are analysed using short descriptive categories. A response may belong to more than one category. The report presents the number of participants represented in each category and provides anonymised examples where they help explain the findings. The automatic records and artefacts are checked before attributing an observed event to participant difficulty.

The free exploration period is analysed separately. The analysis describes the topic selected by each participant, the questions asked, any clarification required, and the observed difficulties, comments, or incidents. Frequencies are shown when several participants explored a similar topic, but no quality score is assigned because the participants could go for different objectives and no new artefacts were generated or imported.

The analysis is primarily descriptive. It does not perform hypothesis tests or treat the study as a comparison between the wizard and manual modelling in DISME. The interpretation compares the observed counts and score distributions with the criteria fixed in Table 6.6, while also reporting the underlying values, incomplete cases, and reasons for failure.

6.4.3 Results

The formal data set contains the 19 participant sessions. The pilot session described in Section 6.3.1 is not included. The participants were students and researchers, and none had used the wizard before the study. The level of familiarity that each participant reported for the conference domain and technology considered in the study is shown in Table 6.8. These characteristics are used only to describe the sample and not to form comparison groups.

Table 6.8: Previous familiarity and experience of the 19 analysed participants.

Characteristic	Distribution
Familiarity with conference processes	None: 10 Little: 3 Some: 5 Much: 1
Experience with low-code or modelling tools	None: 1 Little: 5 Some: 11 Much: 2
Use of generative AI	Weekly: 3 Daily: 16
Experience with DISME	None: 17 Much: 2

All 19 analysed sessions contain complete SUS responses, the six NASA-TLX ratings, and responses to the five study-specific statements. An observed common-workflow duration is also available for every session.

Complete initial and confirmed artefact sets were available for all 19 analysed sessions.

Task completion was documented separately from the assistance provided to participants for each task. The number of participants who completed each task without help, completed with assistance, or failed to complete is presented in Figure 6.1.

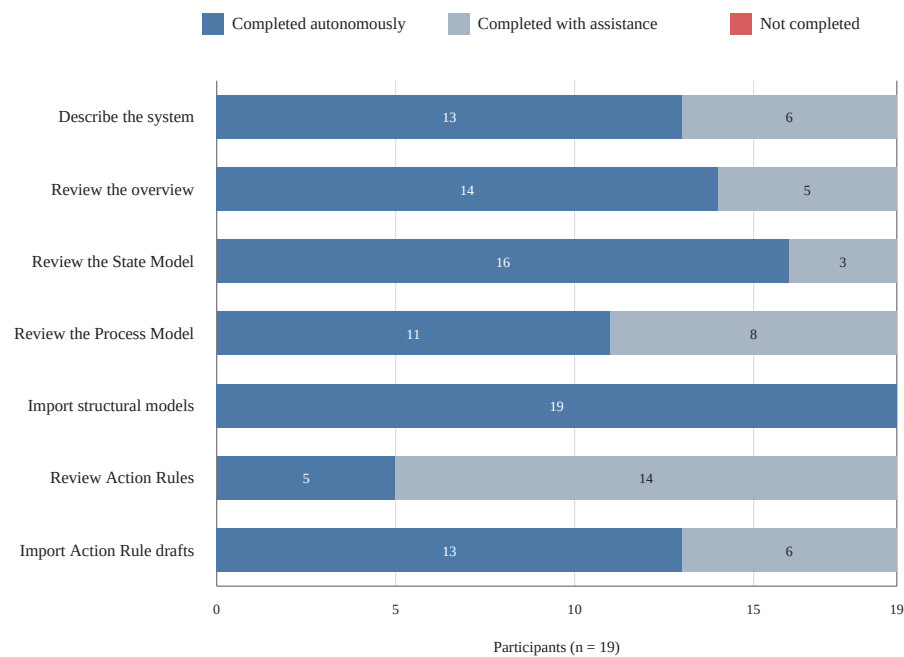


Figure 6.1: Task completion status for the seven assessed tasks.

The first five tasks were completed by the 19 analysed participants. Nineteen participants completed the review and import of the Action Rules. For the review task, 5 participants completed it autonomously and 14 completed it with assistance. For the import of the Action Rule drafts, 13 participants completed it autonomously and 6 completed it with assistance.

Table 6.9 presents the types of assistance recorded for each task. The categories are not mutually exclusive because the same task may require more than one type of assistance.

Table 6.9: Assistance recorded for each participant task.

Task	Procedural	Semantic	Technical
Describe the system	1	6	0
Review the overview	0	5	0
Review the State Model	0	3	0
Review the Process Model	6	8	0
Import the structural models	0	0	0
Review the Action Rules	3	13	3
Import the Action Rule drafts	0	6	0

At session level, procedural assistance existed in 10 of the 19 sessions (52.6%), semantic assistance was recorded in all 19 sessions (100%), and technical assistance existed in 3 of the 19 sessions (15.8%). These values show whether each type of assistance occurred at least once during a session and do not represent the number or duration of the interventions.

The observed common-workflow duration was available for all 19 analysed sessions. As shown in Table 6.10, the mean observed duration was 31.6 minutes and the median was 32 minutes.

The remaining workflow time was calculated by subtracting the recorded duration of the planned moderator operations from the total duration. This period, which includes the participant's use of the wizard and the time spent waiting for its responses, had a mean duration of 30.3 minutes. The planned operations conducted by the moderator in DISME required between 1 and 2 minutes. Technical assistance occurred in 3 sessions, but these brief interventions did not have separately recorded start and end times. Their duration therefore remains included in the remaining workflow time, while their occurrence is reported through the assistance results.

Table 6.10: Duration of the common workflow in decimal minutes.

Measure	n	Mean	SD	Median	Range
Common workflows	19	31.6	1.9	32	28–34
Remaining workflow time	19	30.3	2.0	30	27–33

The session records and observation forms also provide information about how participants interacted with the wizard. The wizard presented 457 questions across the 19 sessions, with a mean of 24.1 questions per session, a median of 25, and a range from 17 to 29. Table 6.11 summarises these interaction counts.

The number of questions was obtained from the automatic session records. Requests for clarification or reformulation and correction requests were obtained from the moderator's observation forms. These counts describe how participants interacted with the workflow and are not individually classified as successful or unsuccessful outcomes.

Table 6.11: Main interactions recorded during the common workflow.

Interaction	Total	Sessions	Source
Questions presented by the wizard	457	19	Session record
Requests for clarification or reformulation	46	17	Observation form
Corrections requested by participants	17	15	Observation form

Artefact quality assessment was performed on all 19 analysed sessions, which had complete initial and confirmed artefact sets. Each result was compared to the 30 quality criteria for the reference scenario. Table 6.12 compares the first complete result presented for review with the result confirmed by the participant.

Table 6.12: Quality of the initial and confirmed artefacts.

Measure	Initial result	Confirmed result
Mean score out of 30	29.1	29.2
Mean rubric score	96.8%	97.3%
Standard deviation in percentage points	4.2	4.1
Median percentage	98.3%	98.3%
Percentage range	83.3–100%	83.3–100%
Sessions with a score of 30	8	8
Sessions meeting the 85% threshold	18	18
Sessions containing critical errors	7	5
Total number of critical errors	11	9

The confirmed quality score increased in 3 sessions, remained unchanged in 16, and did not decrease in any session. Two participants (P01 and P02) had an improvement of 1 point, while participant P20 had an improvement of 0.5 points. P07 remained below the predefined quality threshold, with 25 out of 30 points, corresponding to 83.3%, in both the initial and confirmed results.

Eleven sessions contained at least one difference in their initial result and could therefore produce an increase in quality. According to the calculation defined in Section 6.4.2, the correction rate was:

$$\frac{3}{11} \times 100 = 27.3\%.$$

The source of each improvement was also examined. The improvements in P01 and P20 followed corrections requested during the participant's review. In P02, the increase followed a technical recovery that restored the missing causal link between the two transactions. Therefore, not all 3 score increases represent a difference identified and corrected autonomously by the participant.

The observation forms recorded 17 correction requests across 15 sessions, while only 3 sessions obtained a higher confirmed quality score. Correction requests and score changes were measured independently. A requested change did not necessarily increase the score when it altered wording or another element that already satisfied the corresponding quality criterion.

The aggregate quality of each phase is shown in Table 6.13. The percentage values were calculated by dividing the total number of points earned in that phase by the total number of possible points that could have been earned for that phase over the 19 analysed sessions.

Table 6.13: Artefact quality by wizard phase.

Phase	Initial result	Confirmed result
Phase 0 — Overview	98.7%	100.0%
Phase 1 — State Model	98.9%	98.9%
Phase 2 — Process Model	97.4%	97.4%
Phases 3/3b — Queries	98.7%	98.7%
Phase 4 — Action Rules	94.5%	95.5%

Functional outcomes were analysed separately from the artefact quality scores. Structural artefacts were successfully imported in all 19 analysed sessions. Nineteen sessions reached the Action Rule import stage, and all 19 imported the rule drafts, allowed them to be opened and activated in DISME, and supported the creation of the required forms.

Table 6.14: Functional outcomes obtained in DISME.

Functional outcome	Achieved	Not achieved	NC
Structural models imported	19	0	0
Action Rule drafts imported	19	0	0
Action Rules opened and activated	19	0	0
Required forms created	19	0	0
Both transactions completed for the same article	17	2	0
Accepted confirmed as the current state and Submitted preserved in the history	17	2	0

The tested functional path was achieved in 17 of the 19 analysed sessions, corresponding to 89.5%. In these sessions, the submission and decision transactions both finished for the same article. The final current state was Accepted, while Submitted remained available in the article's history.

P07 and P08 imported the structural models and Action Rule drafts, and the required forms were created. However, the complete execution path was not achieved. In P07, critical errors were recorded for three Action Rule criteria (R25, R27, and R29) and for the process completion criterion (R17). The submission and decision sequence did not reach the expected result. In P08, a condition in the submission rule caused the transaction to roll back before the tested path could finish.

Table 6.15 summarises the 3 sessions that passed the execution checks even with the remaining critical errors.

Quality scores and critical errors were recorded separately. A criterion could receive partial credit when some requirements were satisfied, even if a critical error remained. A single error could also affect more than one criterion.

The rubric and session-level results are provided in Appendix B, Tables B.1 and B.2.

Table 6.15: Critical errors in sessions that passed the execution checks.

Session and criteria	Remaining error	Limit of the execution checks
P13 R17, R30	The decision rule restarts submission instead of ending the process.	Completing both transactions did not prove that the process ended.
P18 R16	The Author executes the decision instead of the Conference Chair.	The test account had both roles.
P19 R05, R20	Content replaces Abstract in both the model and the form.	Both artefacts agree, but use the wrong business concept.

These functional checks concern the behaviour of the generated and confirmed artefacts. The required configuration and execution operations in DISME were performed by the moderator, so these results do not measure the participants' ability to operate DISME.

All 19 analysed participants provided complete SUS responses. The mean SUS score was 75.53, with a standard deviation of 8.88, a median of 75, and scores ranging from 60 to 90. So, the planned criterion of a mean SUS score of at least 70 was met. In addition, 15 out of the 19 SUS ratings individually were 70 or higher. SUS values are scores on the standard 0–100 scale and are not interpreted as percentages.

Perceived workload during the common workflow was evaluated through the six NASA-TLX dimensions. Table 6.16 presents the descriptive results. Higher ratings indicate greater perceived workload for mental demand, physical demand, temporal demand, effort, and frustration. For performance, a higher rating indicates that the participant perceived their performance as more successful.

Table 6.16: NASA-TLX ratings for the common workflow.

Dimension	Mean	SD	Median	Range
Mental demand	63.7	18.2	60.0	35–95
Physical demand	12.4	6.9	10.0	5–30
Temporal demand	79.2	11.6	80.0	50–100
Performance	82.9	7.3	85.0	70–95
Effort	64.5	15.9	70.0	30–85
Frustration	17.4	8.7	15.0	5–35

Temporal demand had the highest mean among the five dimensions in which a higher value indicates greater workload. Physical demand and frustration had the lowest means. The mean performance rating was 82.9 under the positive orientation used by the questionnaire.

Table 6.17 presents the responses to the five study-specific statements. In this table, the distribution column reports for each question the number of answers from 1 through 5. A positive response is a rating of 4 or 5. The planned criterion required a median of at least 4 and at least 14 positive responses.

The planned criterion was achieved for the support provided by the review screens and

Table 6.17: Responses to the study-specific statements.

Statement	Ratings 1–5	Mean	Median	4 or 5	Criterion
The wizard’s questions were clear	0/0/7/8/4	3.84	4.0	12 (63.2%)	No
The review screens supported verification	0/0/3/9/7	4.21	4.0	16 (84.2%)	Yes
The participant could identify and correct incorrect information	0/4/3/6/6	3.74	4.0	12 (63.2%)	No
The participant understood what would be imported	0/2/7/7/3	3.58	4.0	10 (52.6%)	No
The wizard was useful for producing a first draft	0/0/1/3/15	4.74	5.0	18 (94.7%)	Yes

for the perceived usefulness of the wizard in producing a first draft. It was not achieved for question clarity, the ability to identify and correct incorrect information, or understanding of what would be imported. Understanding of the imported content had the lowest mean and the smallest number of positive responses, while perceived usefulness had the highest mean and the largest number of positive responses.

All 19 analysed participants reported that TeamViewer did not interfere with their experience. This response is used only to describe the perceived effect of remote access and is not included in the SUS, NASA-TLX, quality, or functional measures.

The open questionnaire responses, observation notes, and free-exploration records were grouped into descriptive categories. These categories are not mutually exclusive, that is, a participant can be counted in more than one category. Each count represents participants for whom at least one explicit example of the theme was recorded, rather than the total number of comments or events. A missing response was not assigned to a category.

Table 6.18 presents the difficulties and needs identified in the records.

Table 6.18: Difficulties and needs identified in the qualitative records.

Theme	Participants	Recorded evidence
Unclear LLM questions or business concepts	8	Questions about task order, process completion, roles, and the expected content of an answer
Understanding or reviewing the Action Rules	7	Difficulties with Phase 4, the behaviour of the rules, their correction, or their representation as blocks
Understanding the generated result, its import, or its connection to DISME	7	Difficulties relating the review screens to the elements subsequently shown in DISME
Need for more familiarisation	6	Requests for more time to understand the wizard, the domain, or the type of result that should be reviewed
Difference between a role and an entity	2	Explicit confusion between the roles Author and Conference Chair and the entities stored by the system
Need for more review time	1	Difficulty verifying or correcting the result within the available time

The suggested improvements were also grouped into non-exclusive categories.

Table 6.19: Improvements suggested by participants.

Suggested improvement	Participants
Clearer interface, progress information, or editing mechanisms	9
More familiarisation or additional time	5
Clearer explanations during review or at the end of the workflow	5
Improvements to questions, response time, or model interpretation	4

Examples of these suggestions included keeping the current context and artefacts visible during the conversation, showing progress through the workflow more clearly, explaining what participants should verify, and making corrections easier to apply. These suggestions were retained as participant feedback and were not converted into additional usability scores.

The free exploration records were kept separate, and this exploration happened after the common workflow ended. It was conversational and did not generate, score, or import new artefacts. A list of topics discussed with the participants is presented in Table 6.20. Some participants explored more than one of the listed topics, so these categories can overlap.

Table 6.20: Topics followed during free exploration.

Free-exploration topic	Participants
Understanding the created system or the wizard better	9
Discussing a system from another domain	8
Extending the conference scenario	7
Exploring the capabilities or limits of the wizard	5

Questions about the existing result included Action Rules, queries, imports, corrections, and information that might be missing. Extensions to the conference scenario included reviewers, additional properties, permissions, notifications, history, and other entities or tasks. Other areas mentioned and discussed by the participants with the wizard included education, football competitions, local events, vehicle sales, logistics, and supermarkets. Some participants also asked about larger systems, the use of the wizard in other domains, preparation for production, or the level of complexity it could support.

Problems occurred with the wizard's responses during free exploration in 4 sessions. Brazilian Portuguese terms appeared in P09 and P10. In P13, the wizard gave a wrong explanation about the relationship between phases, and in P15 it was given a wrong interpretation of the intended topic. These issues occurred during free exploration, after the common workflow and the NASA-TLX assessment had been completed. They therefore did not alter the recorded common-workflow artefacts, task completion, duration, quality scores, functional outcomes, or NASA-TLX ratings. However, the SUS and study-specific questionnaire were completed after free exploration, so participants' experiences during this stage may have influenced those responses.

The observed results were compared with the criteria defined before formal data collection in Table 6.6. Table 6.21 presents the outcome of each comparison. These criteria are reported without introducing a new combined threshold after observing the data.

Table 6.21: Results of the predefined interpretation criteria.

Criterion	Planned reference	Observed result	Outcome
Strict session criterion	All conditions satisfied in the same session	0 of 19 sessions	Not met
Overall feasibility	At least 16 strict sessions and no mixed participant data	0 strict sessions; no mixed participant data	Not met
Review does not worsen the result	At least 18 sessions	19 of 19 sessions	Met
Correction of initial differences	At least 70% of comparable sessions containing a correctable difference	3 of 11 sessions (27.3%)	Not met
Perceived usability	Mean SUS score of at least 70	Mean of 75.53 from 19 valid scores	Met
Study-specific perceptions	The condition is satisfied for each of the five statements	The condition was satisfied for 2 of the 5 statements	Not met overall
Security and record integrity	No silent invalid import, false functional success, or mixed participant data	No silent invalid import, mixed participant data, or false functional success was recorded	Met

No session satisfied every condition of the strict session criterion. In particular, all 19 analysed sessions contained at least one semantic intervention, which alone prevented them from satisfying this combined criterion. This result does not mean that no participant completed the workflow: all 19 analysed participants completed the seven wizard tasks, with 18 artefact sets meeting the 85% quality threshold and 17 sessions completing the tested functional path.

6.5 Discussion

This section interprets the results in relation to the study goals and then discusses the main threats to validity.

6.5.1 Interpretation of Results

The predefined overall-feasibility criterion was not met. This criterion required at least 16 sessions to satisfy all conditions of the strict session criterion, but no session satisfied every condition. In particular, at least one semantic intervention was recorded in every analysed session. This result means that the study does not support fully autonomous use of the wizard under the defined conditions. However, the individual measures provide evidence

about which parts of the workflow were feasible and where participants continued to require support.

The technical results support the feasibility of generating and using an initial specification for the reduced conference scenario. Structural models were imported in all 19 analysed sessions, and all 19 sessions reached the import and activation of the Action Rules. The tested functional path was achieved in 17 sessions. In these cases, both transactions were completed for the same article and the expected current and historical states were verified. However, P07 and P08 show that successful import does not guarantee correct runtime behaviour. The technical path therefore worked in most sessions, but not in every case. Since the configuration and execution operations in DISME were performed by the moderator, these results concern the generated artefacts and their integration rather than the participants' ability to operate DISME.

All 19 participants were able to complete the seven wizard tasks. This shows that the accompanied workflow can be completed within the session. The mean length of the 19 common workflows was 31.6 minutes. The durations ranged from 28 to 34 minutes, leaving little time within the 35-minute target in some sessions. In addition, at least one semantic intervention was recorded in all 19 sessions, and requests for clarification or reformulation occurred in 17 sessions. Assistance was particularly frequent during the review of the Process Model and Action Rules. These results show that participants completed the workflow in the available time when support was provided, but they do not state that the completion is possible without any assistance.

The time and assistance results are consistent with the perceived workload. Temporal demand had the highest mean among the five dimensions, and higher ratings reveal greater workload. In addition to this, both mental demand and effort also had relatively high mean ratings. Frustration had a mean of 17.4, while perceived performance had a mean of 82.9. This combination suggests that participants generally believed they were progressing towards the requested result, but experienced pressure from the amount of information and the limited time available. The results show that all 19 analysed participants completed the accompanied workflow within the defined period.

The 10-minute familiarisation period may also have limited the participants' ability to understand the workflow before the measured tasks began. A longer practice period, followed by more time for reviewing the generated models, might reduce the need for assistance. However, this was not evaluated in the present study, and it cannot be concluded from these results that additional training would lead to autonomous use.

According to the scenario-specific rubric, the initial artefacts were already close to the reference specification. The initial results had a mean quality of 96.8%, while the confirmed results had a mean of 97.3%. Eighteen of the 19 complete artefact sets met or exceeded the predefined 85% threshold in both versions. No decrease in the rubric score was observed in any session.

These percentages are mean scores under the 30-criterion rubric for the reduced conference scenario. They do not represent the proportion of entirely correct artefact sets

or a general accuracy rate for the LLM. A high rubric score can coexist with a critical error because each criterion contributes only part of the total score.

However, the review produced a higher score in only 3 of the 11 sessions that contained a correctable difference. Furthermore, 1 of these improvements followed a technical recovery rather than a difference identified autonomously by the participant. This result must be considered together with the high quality of the initial artefacts. The initial mean was 96.8%, 8 initial results already received the maximum score, and several requested corrections concerned wording or elements that already satisfied the rubric. There was therefore limited room for measurable improvement in several sessions. Nevertheless, 8 of the 11 sessions containing a correctable difference did not obtain a higher confirmed score, which shows that the review did not consistently lead participants to correct the remaining differences. The number of critical errors decreased from 11 to 9, but 5 confirmed results still contained at least one critical error. Phase 4 had the lowest aggregate quality and required the most semantic assistance, which identifies the Action Rules as the main difficulty in the evaluated workflow. A larger or more complex scenario could provide more opportunities for differences and corrections, but this was not evaluated in the present study.

Participants' perceptions provide a similar result. The mean SUS score of 75.53 satisfied the predefined usability criterion. A total of 16 of the 19 participants also gave a positive evaluation of the review screens and nearly all participants, 18 of the 19, indicated that they found the wizard helpful in producing an initial draft. The planned criteria were not met for question clarity, the ability to identify and correct information, or understanding of what would be imported. Only 10 of the 19 participants gave a positive response about understanding the imported content. Participants can therefore perceive the wizard as useful while still having difficulty understanding some of its questions, generated artefacts, and connection to DISME.

The qualitative records help explain these results. Participants reported difficulties with some LLM questions and business concepts, the interpretation of the Action Rules, and the relationship between the wizard's review screens and the elements later shown in DISME. Suggestions focused on clearer progress information, easier editing, more explanation during review, and additional familiarisation. These observations indicate that the main remaining difficulty is not only the generation of an initial specification, but also supporting users in understanding, verifying, and correcting that specification before import.

Free exploration showed that participants were interested in understanding the generated system, extending the conference scenario, and discussing systems from other domains. This provides examples of the questions and directions that participants considered relevant after completing the common workflow. However, free exploration did not have a common target, did not produce scored artefacts, and did not test import or execution.

6.5.2 Threats to Validity

The threats are organised as construct, internal, external, and conclusion validity, following the categories described by Jedlitschka et al. [53].

6.5.2.1 Construct validity

Task completion, artefact quality, functional execution, workload, and participant perception measure different parts of the workflow and were therefore analysed separately. This quality rubric is not a validated general measure of artefact quality. A high aggregate score can mask individual errors because all criteria have equal weight. Beyond that, every first version had a high score, so the opportunity to measure the improvement between the first and the final reviewed version was limited. Critical errors were therefore reported separately. The assistance categories were also defined for this study: a brief clarification and a detailed explanation both count as semantic assistance. Consequently, the strict criterion indicates whether semantic support occurred, but not its extent. SUS, NASA-TLX, and the study-specific statements describe perceptions rather than artefact correctness or productivity.

6.5.2.2 Internal validity

The LLM produced different questions and artefacts from each participant's information, which may have created paths with different difficulty. The moderator may also have influenced behaviour through explanations, and the researcher designed the wizard, conducted the sessions, and applied the quality rubric. The findings therefore represent accompanied use and may be affected by the researcher's presence and judgement. To reduce other sources of variation, all participants used the same wizard version, configuration, materials, environment, familiarisation, and task order. Automatic records, initial and confirmed artefacts, incomplete outcomes, and moderator interventions were preserved, while a clean baseline and separate session directories reduced the risk of mixed data.

6.5.2.3 External validity

The 19 analysed participants were computer science students and researchers recruited by convenience. All were new to the wizard, but most were familiar with software and generative AI; the sample is therefore not representative of all potential users. The study also used one reduced conference scenario containing one main entity, two roles, and two transactions. It did not evaluate larger systems, relationships between entities, complex queries, or every type of Action Rule. Participants received the business facts in a script, and the moderator performed the configuration and execution operations in DISME. The results therefore concern this accompanied workflow and cannot demonstrate independent use or correct generation in other domains.

6.5.2.4 Conclusion validity

The study used one condition, a convenience sample, and descriptive analysis. It did not compare the wizard with manual modelling or another tool and cannot establish statistical superiority or causal effects. The interpretation thresholds were defined for this study and are not universal pass or fail values. One of the 20 conducted sessions was excluded because the moderator stopped the workflow at the 35-minute target instead of allowing it to continue. The exclusion was based on this procedural deviation, not on the participant's result, but it was made after data collection and reduced the analysed sample to 19 sessions. All 19 analysed sessions contributed to the task, time, questionnaire, workload, and quality results, while the correction analysis used the 11 sessions containing a correctable difference. Missing results were not estimated, and the participant session remained the unit of analysis. The qualitative categories were non-exclusive and assigned by the same researcher, so their frequencies describe the recorded evidence rather than a wider population.

6.6 Summary

This chapter evaluated whether people who had not used the wizard before could follow its workflow to create and review an initial specification for a reduced conference system. It also examined whether the resulting artefacts could be imported and executed in DISME.

All 19 analysed participants completed the seven wizard tasks, the artefacts generally matched the reference specification, and 17 of the 19 sessions completed the tested execution path. The usability criterion was met, and most participants considered the wizard useful for producing a first draft. However, every session required some semantic assistance, and participants did not consistently identify and correct the remaining differences. The main difficulties concerned some interview questions, the Action Rules, and understanding what would be imported.

The study therefore supports the use of the wizard as an accompanied approach for producing an initial DISME specification in the evaluated scenario. It does not demonstrate fully autonomous use, or operation in larger systems and other domains. These results identify the points that should be improved and provide the basis for the conclusions and future work presented in the next chapter.

CONCLUSIONS

This chapter returns to the objective introduced in Chapter 1 and considers what the implemented solution and its evaluation allow us to conclude. It also identifies the next steps for improving the wizard and evaluating it under broader conditions.

7.1 Conclusions and Contributions

Creating an organisational system in DISME does not require conventional programming, but it requires knowledge of how the platform represents entities, properties, roles, processes, transactions, and Action Rules. This dissertation investigated whether an LLM-assisted approach could help users move from a natural-language description to an initial DISME specification while allowing them to review and correct the generated artefacts before import and execution.

The resulting wizard organises this process as a sequence of interviews and review stages. Information confirmed in one phase becomes context for the phases that follow, allowing the specification to be built gradually instead of generated from a single prompt. Structural artefacts are represented in a format that can be checked against DISME's types, identifiers, and references. Action Rules follow a separate path: the wizard produces a structured representation of their behaviour, checks it against the confirmed models and supported operations, and uses the Blockly Bridge to convert it into Blockly XML. The Blockly Bridge was developed in this work and reuses the block definitions and code generators already available in DISME. The confirmed artefacts are then imported into DISME, with the Action Rules kept as drafts until they are inspected and activated.

The Systematic Literature Review informed this design by examining how LLMs have been integrated with low-code and model-driven environments. Its findings supported the use of structured intermediate representations, automated checks, and human review. Based on this foundation, the main technical contribution is a pipeline that connects natural-language interviews to the generation, review, validation, and import of an initial DISME specification. The pipeline does not rely on the LLM alone to produce content that can be used directly by the platform. The user remains responsible for confirming that

the result represents the intended business system.

The participant study evaluated this workflow using a reduced conference management scenario. All 19 analysed participants completed the seven assessed wizard tasks, and all sessions reached the functional verification stage. The structural models and Action Rule drafts were imported in every session. 18 of the 19 confirmed artefact sets met the predefined quality threshold, and no confirmed result received a lower quality score than its initial version. The tested functional path was completed in 17 sessions. These results show that the pipeline could produce artefacts that were accepted by the import mechanisms and supported the defined execution path in most of the evaluated sessions. These results are consistent with the distinction between structural validity and semantic correctness discussed in Chapter 3, since some artefacts passed the execution checks but still contained critical errors.

The usability criterion was met, and 18 participants considered the wizard useful for producing an initial specification. At the same time, every analysed session required at least one semantic intervention, so no session met the predefined strict success criterion. Participants frequently needed clarification when interpreting process concepts, reviewing Action Rules, or understanding how the reviewed content would appear in DISME. Review did not consistently lead them to identify and correct the differences that remained in the generated artefacts. This suggests that the human review discussed in the literature also needs clear explanations that help users understand and check the generated behaviour. The study therefore provides evidence for accompanied use of the wizard. It also evaluates one reduced scenario rather than the creation of a complete production system or every type of model supported by DISME.

Within these conditions, the main objective of the dissertation was achieved. The implemented workflow can turn a natural-language description into an initial specification that users can review and that can be imported into DISME. In most sessions, the resulting artefacts also supported the tested runtime path. For users who are unfamiliar with DISME, the wizard provides a starting point for expressing a business system without first creating every element through the platform's modelling interfaces. The evaluation also identifies where that support is still incomplete, particularly in the explanation and review of generated behaviour.

7.2 Future Work

The first priority is to improve how the wizard explains questions and generated artefacts. Interview questions could state more clearly which business concept is being discussed and what information is expected. During review, the interface could keep the relevant answers and model elements visible, show how a requested correction changed the result, and distinguish the elements that will be imported. Action Rules need particular attention because they combine business behaviour with a representation that was unfamiliar to many participants. A closer connection between the natural-language explanation, the

structured actions, and the resulting blocks could make the rules easier to verify. More direct editing mechanisms could also reduce the need to express every correction through another conversation. The critical errors observed in the study should also guide further improvements to rule generation and validation.

After these changes, the study could be repeated with a longer familiarisation period and more time for review. Participants could first complete a separate practice workflow and then use the wizard with less moderator involvement. This would make it possible to examine whether additional training and clearer guidance reduce semantic assistance and perceived time pressure. A later study could compare the wizard with direct modelling in DISME, using the same scenario and measuring completion time, artefact quality, workload, usability, and required assistance.

The solution should also be evaluated with larger specifications and other business domains. These cases should include several related entities, longer processes, more queries, and a wider range of Action Rules so that the current generation and validation mechanisms are tested with more dependencies. Participants with different technical and domain backgrounds, together with more than one evaluator for semantic quality, would provide evidence beyond the conditions of the present study. Alternative ways of answering the interviews, including spoken input, could also be explored once the questions and review process have been improved.

BIBLIOGRAPHY

- [1] J. M. Lourenço. *The NOVAtesis L^AT_EX Template User's Manual*. NOVA University Lisbon, 2021. URL: <https://github.com/joaomlourenco/novathesis/raw/main/template.pdf> (cit. on p. i).
- [2] The L^AT_EX Project team. *L^AT_EX – A document preparation system*. 1985. URL: <https://www.latex-project.org> (cit. on p. v).
- [3] J. M. Lourenço. *The aidisclose package: Generative AI disclosure checklist and statements*. Version 1.13.0. 2025. URL: <https://github.com/joaomlourenco/aidisclose> (cit. on p. v).
- [4] Y. Suchikova et al. “GAIDeT (Generative AI Delegation Taxonomy): A Taxonomy for Humans to Delegate Tasks to Generative Artificial Intelligence in Scientific Research and Publishing”. In: *Accountability in Research* (2025), pp. 1–27. DOI: [10.1080/08989621.2025.2544331](https://doi.org/10.1080/08989621.2025.2544331) (cit. on p. v).
- [5] C. Richardson and J. R. Rymer. *New Development Platforms Emerge For Customer-Facing Applications*. Report. Forrester Research, 2014-06. URL: <https://www.forrester.com/report/RES113411> (cit. on pp. 1, 6, 110).
- [6] V. Freitas et al. “The DISME low-code platform – from simple diagram creation to system execution”. In: *Proceedings of the 22nd CIAO! Doctoral Consortium, and Enterprise Engineering Working Conference Forum 2022*. Vol. 3388. CEUR Workshop Proceedings. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3388/paper4.pdf> (cit. on pp. 1, 8–12, 15).
- [7] J. L. Dietz and H. B. Mulder. *Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation*. Springer International Publishing, 2020. DOI: [10.1007/978-3-030-38854-6](https://doi.org/10.1007/978-3-030-38854-6) (cit. on pp. 1, 7, 9).
- [8] V. H. S. Freitas. “Extension of Action Rule Grammar and Implementation of Processing Engine of a DEMO Based Low-Code Platform”. MA thesis. Universidade da Madeira, 2023 (cit. on pp. 1, 2, 9–12, 15, 22, 23, 164).

-
- [9] N. Hagel, N. Hili, and D. Schwab. “Turning Low-Code Development Platforms into True No-Code with LLMs”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. MODELS Companion ’24. Linz, Austria: Association for Computing Machinery, 2024, pp. 876–885. ISBN: 9798400706226. DOI: [10.1145/3652620.3688334](https://doi.org/10.1145/3652620.3688334). URL: <https://doi.org/10.1145/3652620.3688334> (cit. on pp. 2, 21, 110, 118, 129).
 - [10] C. Tinnes, A. Welter, and S. Apel. “Software Model Evolution with Large Language Models: Experiments on Simulated, Public, and Industrial Datasets”. In: *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*. IEEE Press, 2025, pp. 950–962. ISBN: 9798331505691. URL: <https://doi.org/10.1109/ICSE55347.2025.00112> (cit. on pp. 2, 17–21, 24, 118, 124, 130–132, 134).
 - [11] K. Peffers et al. “A Design Science Research Methodology for Information Systems Research”. In: *Journal of Management Information Systems* 24.3 (2007-12), pp. 45–77. ISSN: 0742-1222. DOI: [10.2753/MIS0742-1222240302](https://doi.org/10.2753/MIS0742-1222240302). URL: <https://doi.org/10.2753/MIS0742-1222240302> (cit. on p. 3).
 - [12] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE 2007-001. Version 2.3. Keele University and Durham University Joint Report, 2007 (cit. on pp. 3, 16, 113).
 - [13] D. C. Schmidt. “Guest Editor’s Introduction: Model-Driven Engineering”. In: *Computer* 39.2 (2006-02), pp. 25–31. ISSN: 0018-9162. DOI: [10.1109/MC.2006.58](https://doi.org/10.1109/MC.2006.58). URL: <https://doi.org/10.1109/MC.2006.58> (cit. on pp. 6, 109, 110).
 - [14] S. Sendall and W. Kozaczynski. “Model transformation: the heart and soul of model-driven software development”. In: *IEEE Software* 20.5 (2003), pp. 42–45. DOI: [10.1109/MS.2003.1231150](https://doi.org/10.1109/MS.2003.1231150) (cit. on pp. 6, 110).
 - [15] A. Bucaioni, A. Cicchetti, and F. Ciccozzi. “Modelling in Low-Code Development: A Multi-Vocal Systematic Review”. In: *Software and Systems Modeling* 21 (2022), pp. 1959–1981. DOI: [10.1007/s10270-021-00964-0](https://doi.org/10.1007/s10270-021-00964-0). URL: <https://doi.org/10.1007/s10270-021-00964-0> (cit. on p. 6).
 - [16] N. Prinz, C. Rentrop, and M. Huber. “Low-Code Development Platforms – A Literature Review”. In: *AMCIS 2021 Proceedings*. 2021-08. URL: https://aisel.aisnet.org/amcis2021/adv_info_systems_general_track/adv_info_systems_general_track/2/ (cit. on p. 6).
 - [17] A. C. Bock and U. Frank. “Low-Code Platform”. In: *Business & Information Systems Engineering* 63.6 (2021-12), pp. 733–740. ISSN: 1867-0202. DOI: [10.1007/s12599-021-00726-8](https://doi.org/10.1007/s12599-021-00726-8). URL: <https://doi.org/10.1007/s12599-021-00726-8> (cit. on p. 6).

- [18] Y. Luo et al. "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective". In: *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ESEM '21. Bari, Italy: Association for Computing Machinery, 2021. ISBN: 9781450386654. DOI: [10.1145/3475716.3475782](https://doi.org/10.1145/3475716.3475782). URL: <https://doi.org/10.1145/3475716.3475782> (cit. on p. 6).
- [19] A. J. Ko et al. "The state of the art in end-user software engineering". In: *ACM Computing Surveys* 43.3 (2011-04). ISSN: 0360-0300. DOI: [10.1145/1922649.1922658](https://doi.org/10.1145/1922649.1922658). URL: <https://doi.org/10.1145/1922649.1922658> (cit. on p. 6).
- [20] M. X. Liu et al. "'What It Wants Me To Say': Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: [10.1145/3544548.3580817](https://doi.org/10.1145/3544548.3580817). URL: <https://doi.org/10.1145/3544548.3580817> (cit. on pp. 7, 14, 19, 118, 127, 130, 132).
- [21] J. L. G. Dietz. *Enterprise Ontology: Theory and Methodology*. Berlin, Heidelberg: Springer, 2006. DOI: [10.1007/3-540-33149-2](https://doi.org/10.1007/3-540-33149-2) (cit. on pp. 7, 8).
- [22] J. L. G. Dietz and J. A. P. Hoogervorst. "Enterprise ontology in enterprise engineering". In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. ACM, 2008, pp. 572–579. DOI: [10.1145/1363686.1363824](https://doi.org/10.1145/1363686.1363824) (cit. on p. 8).
- [23] J. W. Yoder and R. E. Johnson. "The Adaptive Object-Model Architectural Style". In: *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*. WICSA 3. NLD: Kluwer, B.V., 2002, pp. 3–27. ISBN: 1402071760 (cit. on pp. 8, 15).
- [24] JGraph. *draw.io: Online Diagram Software*. Accessed 28 September 2026. 2024. URL: <https://www.draw.io/> (cit. on p. 11).
- [25] Form.io. *Form.io Documentation*. Accessed 28 September 2026. 2024. URL: <https://help.form.io/> (cit. on p. 11).
- [26] A. Fan et al. "Large Language Models for Software Engineering: Survey and Open Problems". In: *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. 2023, pp. 31–53. DOI: [10.1109/ICSE-FoSE59343.2023.000008](https://doi.org/10.1109/ICSE-FoSE59343.2023.000008) (cit. on pp. 14, 112).
- [27] D. Zan et al. "Large Language Models Meet NL2Code: A Survey". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2023, pp. 7443–7464. DOI: [10.18653/v1/2023.acl-long.411](https://doi.org/10.18653/v1/2023.acl-long.411). URL: <https://aclanthology.org/2023.acl-long.411/> (cit. on p. 14).

-
- [28] A. Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964 (cit. on pp. 14, 111).
- [29] T. B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 1877–1901. URL: <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html> (cit. on p. 14).
- [30] A. Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *International Conference on Learning Representations*. 2020. URL: <https://openreview.net/forum?id=rygGQyrFvH> (cit. on p. 14).
- [31] A. Wright et al. *JSON Schema: A Media Type for Describing JSON Documents*. Draft 2020-12. 2022. URL: <https://json-schema.org/draft/2020-12/json-schema-core.html> (cit. on p. 14).
- [32] S. Geng et al. “Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, 2023, pp. 10932–10952. DOI: 10.18653/v1/2023.emnlp-main.674. URL: <https://aclanthology.org/2023.emnlp-main.674/> (cit. on p. 14).
- [33] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010. ISBN: 978-0321712943 (cit. on p. 14).
- [34] OutSystems. *Generative AI App Builder: Create Apps in Minutes with Mentor*. Accessed 25 September 2026. 2026. URL: <https://www.outsystems.com/generative-ai-app-builder> (cit. on p. 15).
- [35] Mendix. *Mendix AI Assistance (Maia)*. Accessed 25 September 2026. 2026. URL: <https://docs.mendix.com/refguide/mendix-ai-assistance/> (cit. on p. 15).
- [36] Microsoft. *Build Apps Through Conversation with Copilot*. Accessed 25 September 2026. 2026. URL: <https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/ai-conversations-create-app> (cit. on p. 15).
- [37] P. Pattnayak and H. Bohra. *Review of Tools for Zero-Code LLM Based Application Development*. 2025. arXiv: 2510.19747. URL: <https://arxiv.org/abs/2510.19747> (cit. on p. 15).
- [38] Y. Wang et al. “Generating SysML Behavior Models via Large Language Models: an Empirical Study”. In: *Proceedings of the 16th International Conference on Internetware*. Internetware ’25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 366–377. ISBN: 9798400719264. DOI: 10.1145/3755881.3755926. URL: <https://doi.org/10.1145/3755881.3755926> (cit. on pp. 17, 20, 21, 24, 112, 118, 124, 127, 130, 131, 133).

- [39] T. Sun et al. “Automated Implementation Scheme for Low-Code Development Based on Large Language Models”. In: *Proceedings of the 2024 4th International Conference on Signal Processing and Communication Technology*. SPCT '24. New York, NY, USA: Association for Computing Machinery, 2025, pp. 237–241. ISBN: 9798400710636. DOI: [10.1145/3712464.3712507](https://doi.org/10.1145/3712464.3712507). URL: <https://doi.org/10.1145/3712464.3712507> (cit. on pp. 17, 19, 21, 117, 124, 129).
- [40] T. Turchi. “FlowPilot: A Generative AI-Driven Visual Language for Computational Thinking Education”. In: *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2024, pp. 353–355. DOI: [10.1109/VL/HCC60511.2024.00046](https://doi.org/10.1109/VL/HCC60511.2024.00046) (cit. on pp. 17, 117, 126, 131).
- [41] J. Xu et al. “LLM4Workflow: An LLM-based Automated Workflow Model Generation Tool”. In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. ASE '24. Sacramento, CA, USA: Association for Computing Machinery, 2024, pp. 2394–2398. ISBN: 9798400712487. DOI: [10.1145/3691620.3695360](https://doi.org/10.1145/3691620.3695360). URL: <https://doi.org/10.1145/3691620.3695360> (cit. on pp. 18, 117, 124, 125).
- [42] E. C. Koh. “Auto-DSM: Using a Large Language Model to generate a Design Structure Matrix”. In: *Natural Language Processing Journal* 9 (2024), p. 100103. ISSN: 2949-7191. DOI: [10.1016/j.nlp.2024.100103](https://doi.org/10.1016/j.nlp.2024.100103). URL: <https://www.sciencedirect.com/science/article/pii/S2949719124000517> (cit. on pp. 18, 19, 117).
- [43] Y. Li et al. “D2: An LLM agent driving end-to-end visual AI modeling in energy platforms”. In: *Energy and AI* 21 (2025), p. 100582. ISSN: 2666-5468. DOI: [10.1016/j.egyai.2025.100582](https://doi.org/10.1016/j.egyai.2025.100582). URL: <https://www.sciencedirect.com/science/article/pii/S2666546825001144> (cit. on pp. 18, 117, 128).
- [44] V. Liventsev et al. “Fully Autonomous Programming with Large Language Models”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Association for Computing Machinery, 2023, pp. 1146–1155. DOI: [10.1145/3583131.3590481](https://doi.org/10.1145/3583131.3590481). URL: <https://doi.org/10.1145/3583131.3590481> (cit. on pp. 18, 20, 21, 111, 118, 125, 130–134).
- [45] A. Grishina et al. “Fully Autonomous Programming Using Iterative Multi-Agent Debugging with Large Language Models”. In: *ACM Transactions on Evolutionary Learning and Optimization* 5.1 (2025-03). ISSN: 2688-299X. DOI: [10.1145/3719351](https://doi.org/10.1145/3719351). URL: <https://doi.org/10.1145/3719351> (cit. on pp. 18, 19, 21, 118, 125, 131–133).
- [46] R. Bavishi et al. “Neurosymbolic repair for low-code formula languages”. In: *Proceedings of the ACM on Programming Languages* 6.OOPSLA2 (2022-10). DOI: [10.1145/3563327](https://doi.org/10.1145/3563327). URL: <https://doi.org/10.1145/3563327> (cit. on pp. 18, 118, 133).

- [47] Z. Zhou et al. "InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs". In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. CHI '25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713941. DOI: [10.1145/3706598.3713905](https://doi.org/10.1145/3706598.3713905). URL: <https://doi.org/10.1145/3706598.3713905> (cit. on pp. 18, 19, 118, 133, 134).
- [48] T. R. Silva and T. T. Hildebrandt. "Human-Centric Hybrid-AI for No-Code Development". In: *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. FSE Companion '25. Clarion Hotel Trondheim, Trondheim, Norway: Association for Computing Machinery, 2025, pp. 1520–1521. ISBN: 9798400712760. DOI: [10.1145/3696630.3731669](https://doi.org/10.1145/3696630.3731669). URL: <https://doi.org/10.1145/3696630.3731669> (cit. on pp. 18, 110, 111, 117, 126, 127, 132–134).
- [49] B. Chen et al. "Accurate and Consistent Graph Model Generation from Text with Large Language Models". In: *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2025, pp. 130–141. DOI: [10.1109/MODELS67397.2025.00018](https://doi.org/10.1109/MODELS67397.2025.00018) (cit. on pp. 19, 118, 125, 126, 128).
- [50] L. Netz, J. Reimer, and B. Rumpe. "Using Grammar Masking to Ensure Syntactic Validity in LLM-based Modeling Tasks". In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. MODELS Companion '24. Linz, Austria: Association for Computing Machinery, 2024, pp. 115–122. ISBN: 9798400706226. DOI: [10.1145/3652620.3687805](https://doi.org/10.1145/3652620.3687805). URL: <https://doi.org/10.1145/3652620.3687805> (cit. on pp. 19, 118, 126, 128, 130, 133).
- [51] N. Malamas et al. "Toward efficient vibe coding: An LLM-based agent for low-code software development". In: *Journal of Computer Languages* 85 (2025), p. 101367. ISSN: 2590-1184. DOI: [10.1016/j.cola.2025.101367](https://www.sciencedirect.com/science/article/pii/S259011842500053X). URL: <https://www.sciencedirect.com/science/article/pii/S259011842500053X> (cit. on pp. 21, 118, 129, 130).
- [52] W. Zhang et al. "Large Language Models in Model-Driven Engineering: A Systematic Mapping Study". In: *Empirical Software Engineering* 32, 3 (2027). DOI: [10.1007/s10664-026-10921-4](https://doi.org/10.1007/s10664-026-10921-4) (cit. on p. 21).
- [53] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. "Reporting Experiments in Software Engineering". In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. Springer, 2008, pp. 201–228 (cit. on pp. 64, 70, 93).
- [54] International Organization for Standardization. *Ergonomics of Human-System Interaction—Part 11: Usability: Definitions and Concepts*. ISO 9241-11:2018. 2018. URL: <https://www.iso.org/standard/63500.html> (cit. on p. 70).
- [55] J. Brooke. "SUS: A "Quick and Dirty" Usability Scale". In: *Usability Evaluation in Industry*. Ed. by P. W. Jordan et al. Taylor & Francis, 1996, pp. 189–194 (cit. on pp. 73, 142).

- [56] A. I. Martins et al. “European Portuguese Validation of the System Usability Scale (SUS)”. In: *Procedia Computer Science* 67 (2015), pp. 293–300. DOI: [10.1016/j.procs.2015.09.273](https://doi.org/10.1016/j.procs.2015.09.273). URL: <https://doi.org/10.1016/j.procs.2015.09.273> (cit. on pp. 73, 142).
- [57] S. G. Hart and L. E. Staveland. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. In: *Human Mental Workload*. Ed. by P. A. Hancock and N. Meshkati. Vol. 52. Advances in Psychology. North-Holland, 1988, pp. 139–183. DOI: [10.1016/S0166-4115\(08\)62386-9](https://doi.org/10.1016/S0166-4115(08)62386-9) (cit. on pp. 73, 141).
- [58] A. Bangor, P. T. Kortum, and J. T. Miller. “An Empirical Evaluation of the System Usability Scale”. In: *International Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594. DOI: [10.1080/10447310802205776](https://doi.org/10.1080/10447310802205776). URL: <https://doi.org/10.1080/10447310802205776> (cit. on p. 74).
- [59] A. Sahay et al. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 171–178. DOI: [10.1109/SEAA51224.2020.00036](https://doi.org/10.1109/SEAA51224.2020.00036) (cit. on pp. 109, 110).
- [60] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. 2nd ed. Synthesis Lectures on Software Engineering. Cham: Springer, 2017. DOI: [10.1007/978-3-031-02549-5](https://doi.org/10.1007/978-3-031-02549-5) (cit. on p. 109).
- [61] X. Hou et al. “Large Language Models for Software Engineering: A Systematic Literature Review”. In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024-12). ISSN: 1049-331X. DOI: [10.1145/3695988](https://doi.org/10.1145/3695988). URL: <https://doi.org/10.1145/3695988> (cit. on pp. 109, 112).
- [62] L. Burgueño et al. “Automation in Model-Driven Engineering: A Look Back, and Ahead”. In: *ACM Transactions on Software Engineering and Methodology* 34.5 (2025-05). ISSN: 1049-331X. DOI: [10.1145/3712008](https://doi.org/10.1145/3712008). URL: <https://doi.org/10.1145/3712008> (cit. on pp. 109, 110, 113, 117).
- [63] J. Cabot. “Positioning of the low-code movement within the field of model-driven engineering”. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS ’20. Virtual Event, Canada: Association for Computing Machinery, 2020. ISBN: 9781450381352. DOI: [10.1145/3417990.3420210](https://doi.org/10.1145/3417990.3420210). URL: <https://doi.org/10.1145/3417990.3420210> (cit. on p. 110).
- [64] K. Czarnecki and S. Helsen. “Feature-based survey of model transformation approaches”. In: *IBM Systems Journal* 45.3 (2006-07), pp. 621–645. ISSN: 0018-8670. DOI: [10.1147/sj.453.0621](https://doi.org/10.1147/sj.453.0621). URL: <https://doi.org/10.1147/sj.453.0621> (cit. on p. 110).

- [65] T. Buchmann, R. Peinl, and F. Schwägerl. “White-box LLM-supported Low-code Engineering: A Vision and First Insights”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. MODELS Companion '24*. Linz, Austria: Association for Computing Machinery, 2024, pp. 556–560. ISBN: 9798400706226. DOI: [10.1145/3652620.3687803](https://doi.org/10.1145/3652620.3687803). URL: <https://doi.org/10.1145/3652620.3687803> (cit. on pp. 111, 118).
- [66] M. Wang et al. “Exploring Influence Factors on LLM Suitability for No-Code Development of End User Applications”. In: *Software: Practice and Experience* 56.1 (2026), pp. 96–118. DOI: [10.1002/spe.70027](https://doi.org/10.1002/spe.70027) (cit. on pp. 111, 118).
- [67] M. Chen et al. “Evaluating Large Language Models Trained on Code”. In: *arXiv preprint arXiv:2107.03374* (2021). URL: <https://arxiv.org/abs/2107.03374> (cit. on p. 111).
- [68] J. Martins, F. Branco, and H. Mamede. “Combining low-code development with ChatGPT to novel no-code approaches: A focus-group study”. In: *Intelligent Systems with Applications* 20 (2023), p. 200289. ISSN: 2667-3053. DOI: [10.1016/j.iswa.2023.200289](https://doi.org/10.1016/j.iswa.2023.200289) (cit. on pp. 112, 117, 130).
- [69] H. Naveed et al. “Model driven engineering for machine learning components: A systematic literature review”. In: *Information and Software Technology* 169, 107423 (2024). DOI: [10.1016/j.infsof.2024.107423](https://doi.org/10.1016/j.infsof.2024.107423). URL: <https://doi.org/10.1016/j.infsof.2024.107423> (cit. on p. 112).
- [70] A. Bucchiarone et al. “Grand challenges in model-driven engineering: an analysis of the state of the research”. In: *Software and Systems Modeling* 19.1 (2020-01), pp. 5–13. ISSN: 1619-1366. DOI: [10.1007/s10270-019-00773-6](https://doi.org/10.1007/s10270-019-00773-6). URL: <https://doi.org/10.1007/s10270-019-00773-6> (cit. on pp. 112, 113).
- [71] J. Di Rocco et al. “On the use of large language models in model-driven engineering”. In: *Software and Systems Modeling* 24.3 (2025), pp. 923–948. DOI: [10.1007/s10270-025-01263-8](https://doi.org/10.1007/s10270-025-01263-8) (cit. on p. 113).
- [72] N. Rao et al. “AI for Low-Code for AI”. In: *Proceedings of the 29th International Conference on Intelligent User Interfaces. IUI '24*. Greenville, SC, USA: Association for Computing Machinery, 2024, pp. 837–852. ISBN: 9798400705083. DOI: [10.1145/3640543.3645203](https://doi.org/10.1145/3640543.3645203). URL: <https://doi.org/10.1145/3640543.3645203> (cit. on pp. 117, 130).
- [73] S.-K. Wang et al. “ChatOps for microservice systems: A low-code approach using service composition and large language models”. In: *Future Generation Computer Systems* 161.C (2024-12), pp. 518–530. ISSN: 0167-739X. DOI: [10.1016/j.future.2024.07.029](https://doi.org/10.1016/j.future.2024.07.029). URL: <https://doi.org/10.1016/j.future.2024.07.029> (cit. on pp. 117, 125).

- [74] Y. Fang and Z. Fan. "Complex Flowchart Text Generation Based on Large Language Models". In: *2025 5th International Conference on Neural Networks, Information and Communication Engineering (NNICE)*. 2025, pp. 737–742. DOI: [10.1109/NNICE64954.2025.11064395](https://doi.org/10.1109/NNICE64954.2025.11064395) (cit. on p. 117).
- [75] N. De Troyer, R. De Croon, and K. Verbert. "Enhancing Accessibility for Non-experts in Low-Code/No-Code Through the Addition of an AI Chatbot". In: *Human-Computer Interaction – INTERACT 2025*. Vol. 16108. Lecture Notes in Computer Science. Cham: Springer, 2026, pp. 203–212. DOI: [10.1007/978-3-032-04999-5_12](https://doi.org/10.1007/978-3-032-04999-5_12) (cit. on p. 117).
- [76] S. Mowzoon and M. Faustini. "Introducing LLMScript: A Turing Complete Prompt Based Scripting Language for LLMs with No External Coding Required". In: *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. 2024, pp. 116–124. DOI: [10.1109/FLLM63129.2024.10852477](https://doi.org/10.1109/FLLM63129.2024.10852477) (cit. on p. 117).
- [77] A. Nour Eldin et al. "Low-code solutions for business process dataflows: From modeling to execution". In: *Information Systems* 135 (2026), p. 102577. ISSN: 0306-4379. DOI: [10.1016/j.is.2025.102577](https://doi.org/10.1016/j.is.2025.102577). URL: <https://www.sciencedirect.com/science/article/pii/S0306437925000614> (cit. on pp. 117, 126).
- [78] Y. Cheng et al. "Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains". In: *ACM Transactions on Software Engineering and Methodology* 33.5 (2024-06). ISSN: 1049-331X. DOI: [10.1145/3638247](https://doi.org/10.1145/3638247). URL: <https://doi.org/10.1145/3638247> (cit. on p. 117).
- [79] F. C. Kunze and A. Fay. "Automated Generation of AML Models for Industrial Plants Using LLM Chat Applications". In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 1–4. DOI: [10.1109/ETFA61755.2024.10710654](https://doi.org/10.1109/ETFA61755.2024.10710654) (cit. on pp. 118, 126).
- [80] K. Deng et al. "Power Grid Model Generation Based on the Tool-Augmented Large Language Model". In: *IEEE Transactions on Power Systems* 40.6 (2025), pp. 5487–5490. DOI: [10.1109/TPWRS.2025.3609049](https://doi.org/10.1109/TPWRS.2025.3609049) (cit. on pp. 118, 127).
- [81] F. He et al. "MetaCloze: a schema-guided automated building metadata model generation system via information extraction". In: *Knowledge-Based Systems* 330.Part B (2025), p. 114596. ISSN: 0950-7051. DOI: [10.1016/j.knosys.2025.114596](https://doi.org/10.1016/j.knosys.2025.114596). URL: <https://www.sciencedirect.com/science/article/pii/S0950705125016351> (cit. on p. 118).
- [82] S. Barat et al. "Constructing Enterprise Digital Twins by Augmenting LLMs with MDE". In: *Proceedings of the 18th Innovations in Software Engineering Conference. ISEC '25*. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400714245. DOI: [10.1145/3717383.3717391](https://doi.org/10.1145/3717383.3717391). URL: <https://doi.org/10.1145/3717383.3717391> (cit. on pp. 118, 125, 134).

- [83] J. N. Kreikemeyer et al. "Using (Not-so) Large Language Models to Generate Simulation Models in a Formal DSL: A Study on Reaction Networks". In: *ACM Transactions on Modeling and Computer Simulation* 35.4 (2025-09). ISSN: 1049-3301. DOI: [10.1145/3733719](https://doi.org/10.1145/3733719). URL: <https://doi.org/10.1145/3733719> (cit. on p. 118).
- [84] J. Zhang and I. Arawjo. "ChainBuddy: An AI-assisted Agent System for Generating LLM Pipelines". In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. CHI '25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713941. DOI: [10.1145/3706598.3714085](https://doi.org/10.1145/3706598.3714085). URL: <https://doi.org/10.1145/3706598.3714085> (cit. on pp. 118, 134).
- [85] D. Vella Zarb, G. Parks, and T. Kipouros. "Program Synthesis with LLM-Predicted Minimal Specialized Grammars". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '25. NH Malaga Hotel, Malaga, Spain: Association for Computing Machinery, 2025, pp. 1072–1080. ISBN: 9798400714658. DOI: [10.1145/3712256.3726430](https://doi.org/10.1145/3712256.3726430). URL: <https://doi.org/10.1145/3712256.3726430> (cit. on pp. 118, 127).
- [86] P. Lewis et al. "Retrieval-augmented generation for knowledge-intensive NLP tasks". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546 (cit. on p. 132).
- [87] B. Shneiderman. "Human-centered AI: ensuring human control while increasing automation". In: *Proceedings of the 5th Workshop on Human Factors in Hypertext*. HUMAN '22. Barcelona, Spain: Association for Computing Machinery, 2022. ISBN: 9781450394017. DOI: [10.1145/3538882.3542790](https://doi.org/10.1145/3538882.3542790). URL: <https://doi.org/10.1145/3538882.3542790> (cit. on p. 132).
- [88] S. Amershi et al. "Guidelines for Human-AI Interaction". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: [10.1145/3290605.3300233](https://doi.org/10.1145/3290605.3300233). URL: <https://doi.org/10.1145/3290605.3300233> (cit. on p. 133).
- [89] E. M. Bender et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. DOI: [10.1145/3442188.3445922](https://doi.org/10.1145/3442188.3445922). URL: <https://doi.org/10.1145/3442188.3445922> (cit. on p. 133).

Large Language Models in Low-Code and Model-Driven Platforms: A Systematic Literature Review

Abstract

Context: Low-code development systems are designed to make it easier to create software applications by lowering the technical barriers associated with traditional coding. In most cases, this is achieved through graphical modelling systems. Large language models (LLMs) have emerged as tools for generating structured artefacts from natural-language input in low-code development. Thus, LLMs challenge the need for users to manually define their systems using traditional modelling paradigms.

Objective: The purpose of this systematic literature review is to analyse the application of Large Language Models in Low-Code and model driven platform systems. The main objective is to catalogue, categorise, and assess how large language models are integrated into these platform systems. Specifically, we seek to understand the current methodologies and intermediate representations employed to produce artefacts such as workflow, rules, and models from natural language prompts.

Method: We conducted a systematic literature review following the guidelines of Kitchenham and Charters. Database searches returned 69 records, of which four were duplicates. The final sample, identified through database searches and snowballing, comprised 35 studies published between 2022 and 2025. We extracted data on integration architectures, generated artefacts, intermediate representations, validation mechanisms, and evaluation results.

Conclusion: The reviewed studies describe approaches that combine LLM generation with intermediate representations, validation, and user review. Some studies report efficiency gains for the tasks evaluated, but semantic errors and the effort required for review and

correction remain concerns. Differences in tasks and evaluation methods limit direct comparisons between approaches.

Keywords: Low-Code Development, Large Language Models, Model-Driven Engineering, Systematic Literature Review, Generative AI.

A.1 Introduction

With the emergence of both Low-Code Development Platforms (LCDPs) [59], and Model-Driven Engineering (MDE) [13], there has been an important evolution in the field of software development. Both paradigms plan to facilitate the process of developing applications by simplifying the complexity of coding, allowing developers and domain experts to develop applications using visual interfaces and high-level models. Although, as noted by [60], historical automation in these types of environments, generally, rely on deterministic transformations and therefore can be very inflexible and difficult to scale.

However, another type of paradigm shift is occurring, as a result of the rapid growth of Generative AI. Large Language Models (LLMs) have demonstrated impressive capabilities for understanding natural language, as well as generating code. Therefore, according to a recent systematic literature review [61], LLMs have moved the focus of software engineering research from simple code completion to more complex generative tasks and therefore, raise questions about how they can best be utilised in structured modelling environments.

Although the possibilities of LLMs may be promising, utilising them in engineering environments encounters many of the same obstacles as they do in general purpose programming. Unlike the relatively flexible nature of standard scripting, utilising LLMs to create structured outputs within MDE is much more restrictive because, unlike simply following syntactic rules, the generated output must conform to the strict metamodels of engineering models. This is emphasised by [62], who state that one of the major challenges facing the utilisation of LLMs in MDE is bridging the gap between probabilistic AI models and the rigid requirements of engineering models. Furthermore, although the field of study concerning the intersection of these two areas is growing rapidly, it remains fragmented and scattered throughout different disciplines.

Therefore, this systematic literature review examines how Large Language Models are being utilised in both Low-Code and model-driven platforms. It identifies the current methods, representations and challenges associated with the automated creation of structured artefacts. Specifically, the review intends to identify, categorise and analyse the approaches that utilise LLMs in these environments. Additionally, the review will examine the methods, representations, and validation techniques employed in creating structured artefacts automatically.

The remainder of this review is organised as follows: Section A.2 introduces the

background and related work. Section A.3 describes the review methodology, including the research questions in Section A.3.1. Section A.4 presents the results, and Section A.5 discusses their implications. Section A.6 addresses threats to validity, and Section A.7 concludes the review.

A.2 Background and Related Work

This section provides details on the main topics addressed by this review: Low-Code Development (LCD), Model-Driven Engineering (MDE), and Large Language Models (LLMs). It also discusses relevant articles and other Systematic Literature Reviews found in the literature to contextualise the present study.

A.2.1 Low-Code and Model-Driven Engineering

The Model-Driven Engineering (MDE) methodology represents a paradigm shift within software engineering where the focus is on models as the primary development artefacts, using them to define how the system works, instead of just describing it [13]. In doing so, MDE seeks to address complexity issues while improving productivity through the generation of executable systems using automated transformations of abstract models to concrete models or other representations. Automation was at the core of MDE, however, it was implemented via strict model transformations that translate abstract specifications into concrete code or other model representations [14].

Similarities exist between MDE and Low-Code Development Platforms (LCDPs), specifically the use of graphical user interfaces, visual metaphors, and configuration tools to rapidly deliver applications with minimal hand coding [59]. Richardson and Rymer [5] emphasise that LCDPs allow non-programmers to create applications without requiring extensive programming knowledge. Additionally, LCDPs can leverage model-driven paradigms under the hood to generate executable applications from visual specifications. Therefore, LCDPs represent an industrial realisation of MDE methodologies [63].

A.2.1.1 Challenges of Automation in MDE

While traditional automation in MDE relies on deterministic transformations, i.e., Model-to-Model (M2M) and Model-to-Text (M2T) transformations defined by strict transformation rules [64], they have been successful for well-defined domains. However, they are typically inflexible and difficult to scale, thus requiring maintenance efforts.

More recently, researchers have demonstrated a growing requirement for more flexible and intent-based automation strategies, as seen in the studies [62], [48]. Research by Hagel et al. [9] reinforces this aspect, identifying that traditional Low-Code platforms still require users to write code in DSLs and understand metamodels, which creates a barrier for them to have a true "No-Code" experience and also does not allow the environment to flexibly interpret vague user input.

A.2.2 Large Language Models in Software Engineering

The emergence of Large Language Models (LLMs) represents a significant step forward in the development of Artificial Intelligence (AI). As a result of advancements in transformer architecture, developed by Vaswani et al. [28], LLMs are able to recognise and utilise large amounts of training data for both text and source code in order to identify numerous patterns and structures within multiple programming languages. Unlike previous deep learning models, LLMs have demonstrated an ability to perform well in few-shot learning and reasoning, and therefore require little to no additional training data to be effective in new areas.

As stated previously, the utilisation of AI in the field of Software Engineering (SE) has progressed significantly. The ability of AI to perform complex generative functions has grown, and LLMs have become a viable option to automate tasks throughout all aspects of the software lifecycle. LLMs have been explored for tasks such as code generation, testing, program repair, and requirements engineering. These models are currently being utilised for a variety of SE-related tasks such as automated coding, automated testing, automated program repair [44], and automated requirements engineering.

While there are many benefits to utilising LLMs for engineering-related tasks, there are also several challenges associated with this method of operation. As mentioned by Buchmann et al. [65], one challenge that exists is the "black-box" nature of LLMs. The authors state that it can be difficult for engineers to understand how the model arrived at a particular piece of code, and thus there are concerns surrounding trust and maintainability. In addition, Wang et al. [66] observed that while LLMs perform very well at creating general-purpose scripting, the performance of LLMs can vary depending on the specific domain and the prompting strategy used. Due to the probabilistic nature of LLMs, they tend to generate code based on the probability of the next token, rather than based on determinism [67]. As a result, LLMs can exhibit "hallucinations," where they create code that is syntactically correct, but functionally incorrect.

A.2.3 The Intersection: AI-Augmented Modelling

With the convergence of Generative Artificial Intelligence (AI), and Model-Driven Engineering (MDE), a new paradigm for engineering is evolving. This paradigm is typically referred to as "AI-Augmented Modelling". Unlike the automation of traditional MDE, which relies on the manual definition of transformation rules, AI-Augmented Modelling utilises Large Language Models (LLMs) to define those transformations based upon either informal or semi-formal input provided to them. This offers a new method to connect natural language requirements with formal models.

The combination of these two disciplines has the potential to result in huge changes.

It was suggested in [48] that LLMs can serve as translation layers, turning ambiguously worded textual descriptions into Domain Specific Languages (DSLs). This would enable modelling to be accessible to users who do not have programming skills, similar to

Low-Code tools, however would allow the user to express their modelling intent in conversational format. In addition, [68] suggested that the integration of a chat-based interface with a low-code platform will allow designers to move past the complexity of the UI interaction, enabling designers to concentrate on the high-level design logic of the design.

However, the integration of these two disciplines presents a new challenge: how to connect the probabilistic nature of LLMs with the deterministic nature of engineering models. Unlike the semantics of natural languages which can be somewhat vague or variable, models in MDE must stick to their metamodels and structural constraints in order to be executable. [38] empirically demonstrated in the generation of Systems Modeling Language (SysML) models that while LLMs can generate valid code structures with a very high degree of syntactic correctness, they can also frequently produce semantic hallucinations. The findings presented by this research support the need for additional validation checks on the generated models by humans, as well as the need for intermediate validation checks to ensure the dependability of the AI-generated artefacts in safety-critical systems.

A.2.4 Related Work

Existing secondary research (Systematic Literature Reviews and Surveys) that are positioned at the interface between AI and Software Engineering were analysed to place this study in the larger body of academic research. The majority of reviews addressing both areas separately do not provide an overall classification of integration techniques for Generative AI in Low-Code environments.

LLMs for Software Engineering: The systematic literature review [61], along with the extensive survey by Fan et al. [26], provided an overview of the use of LLMs in software engineering in general. These works covered in detail many of the software engineering tasks, including code generation, test synthesis, and repair of programs.

These reviews address software engineering more broadly. The present review focuses on the integration of LLMs into Low-Code and model-driven platforms, including intermediate representations and the validation of generated models.

MDE for Machine Learning: [69] performed a review of how model-driven engineering could be applied to the creation of Machine Learning (ML) components (named "MDE4ML"), utilising modelling concepts to manage ML complexity. Unlike the work presented in [69], which explores applying engineering models to AI, our review investigates the inverse relationship of "AI for MDE" (utilising AI to create software engineering models), which has a different set of validation and correctness issues.

LLMs can generate models that are syntactically valid but do not meet the intended requirements. Their outputs therefore need to be checked for both syntactic and semantic correctness.

Automation in MDE: Bucchiarone et al. [70] pointed out that the complexity of

formal languages and usability issues are still significant barriers to the adoption of MDE. Addressing these issues, the article [62] has documented how MDE has evolved towards greater use of automation by identifying LLMs as a key area of advancement in the field of MDE and suggesting that hybrid methods using both symbolic processing techniques and generative models are an appropriate way to move forward.

LLMs in MDE: Di Rocco et al. [71] conducted a systematic review of LLM applications in MDE and the use of MDE to support LLMs. They organised the reviewed work by modelling task and discussed technical integration, evaluation challenges, and future research directions.

Current Fragmented Landscape: Even though research about the inclusion of Generative AI in Low-Code is growing, it remains fragmented. There are many studies centred on how developers can assure the usability and performance of AI-assisted programming tools. Bucchiarone et al. [70] and Burgueño et al. [62] provide examples of how the technical aspects of current modelling languages are a major barrier in terms of flexibility and complexity when adopting Visual Low-Code Platforms. Therefore, there is a need to develop a next generation of flexible and automated systems to address these problems.

This SLR provides a systematic categorisation of the technical methods used to integrate LLMs into Low-Code and MDE systems. Specifically, we seek to classify how structured intermediate artefacts are created using LLMs and the mechanisms used to check syntactic and semantic correctness.

A.3 Methodology

This SLR was performed using the guidelines for performing systematic literature reviews in software engineering developed by Kitchenham and Charters [12]. The method is intended to provide a reproducible approach to the review process, and the entire review process is supported by Parsifal, a web based tool to support researchers in the planning, execution, and reporting of systematic literature reviews.

The review process is organised into three major components: planning, conducting, and documenting. The planning component identified the research objectives, defined the research questions, search strategy, and the inclusion/exclusion criteria, which were used to identify the most relevant studies. The conducting component implemented the search strategy across the selected digital libraries, filtered the results according to the previously determined criteria, and systematically extracted data from the selected studies. Finally, the documentation component analysed and synthesised the extracted data to determine answers to the research questions and report the findings as they appear in this document.

Below are the subsections that detail the protocol applied in this study, which begin with an identification of the research objectives and the questions that guided this investigation.

A.3.1 Goals and Research Questions

This review is primarily focused on analysing the current state of the art concerning the combination of LLMs with Low-Code and Model Driven Engineering. We have identified that our research questions will focus upon identifying how LLMs create artefacts/intermediate representations, the ones that are utilised for bridging a language model’s output and an executable model’s input, and validation techniques that can be used to confirm that the generated model is correct.

To organise the investigation and determine the scope of the literature review, we employed the PICOC framework (Population, Intervention, Comparison, Outcome, Context) as shown in Table A.1.

Table A.1: PICOC Analysis.

Criteria	Description
Population	Studies, frameworks, and platforms related to Low-Code and model-driven development.
Intervention	The use of Large Language Models to generate, assist, or validate structured artefacts such as rules, workflows, and models.
Comparison	Traditional rule/model generation methods without AI assistance, or non-LLM approaches.
Outcome	Identified methods, intermediate representations, validation techniques, and challenges in integrating LLMs.
Context	Low-code and model-driven development environments, practical application within the DISME platform.

Based on the results of our search scope, we defined four Research Questions (RQs), which are used to define the data extraction and synthesis process:

- **RQ1: How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?** This question will investigate the different design and technical methods that authors have employed to include LLMs in a Low-Code environment. It analyses the architectural coupling, that is, how the model is connected to the environment. In addition, it explores the interaction between the LLM’s generative capability and the modelling tool’s determinism.
- **RQ2: What types of intermediate or structured representations are used between natural language inputs and executable artefacts?** Since direct generation of a model is very difficult at times, we want to determine how some intermediate representations (like JSON, DSLs, or graph structures) help separate the probabilistic representation of LLMs from the determinism required by most platforms.
- **RQ3: What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?** Since LLMs can produce false information due to hallucination,

understanding the current validation techniques (such as human-in-the-loop validation, compiler-based validation, etc.) that are being utilised to ensure the model's reliability is critical.

- **RQ4: How are these approaches evaluated, and what benefits or limitations are reported in AI-assisted Low-Code development?** We want to measure the level of maturity within this area through empirical data on the improvements in efficiency, usability, and issues associated with this approach versus traditional old methods.

A.3.2 Search Strategy and Study Selection

We have developed a search strategy for identifying all relevant studies by focusing on the intersection of three primary concepts: (1) the paradigm of software development (Low-Code or Model Driven); (2) the underlying enabling technology (large language models); (3) the outcome we want from using this technology (automation or generation).

The search string is designed to combine each concept with their respective synonyms and alternative spellings in order to maximise the retrieval of all studies.

("Low-Code" OR "No-Code" OR "Model-Driven" OR "MDE") AND ("Large Language Model" OR "LLM" OR "Generative AI" OR "GPT") AND ("Generation" OR "Automation" OR "Modelling")

The search was conducted on 5th December 2025 across four major digital libraries recognised for their relevance in software engineering research: ACM Digital Library¹, IEEE Xplore², ScienceDirect³, and Springer Link⁴. This process resulted in a total of 69 initial articles. Figure A.1 provides a breakdown of the number of results retrieved from each database.

¹<https://dl.acm.org/>

²<https://ieeexplore.ieee.org/>

³<https://www.sciencedirect.com/>

⁴<https://link.springer.com/>

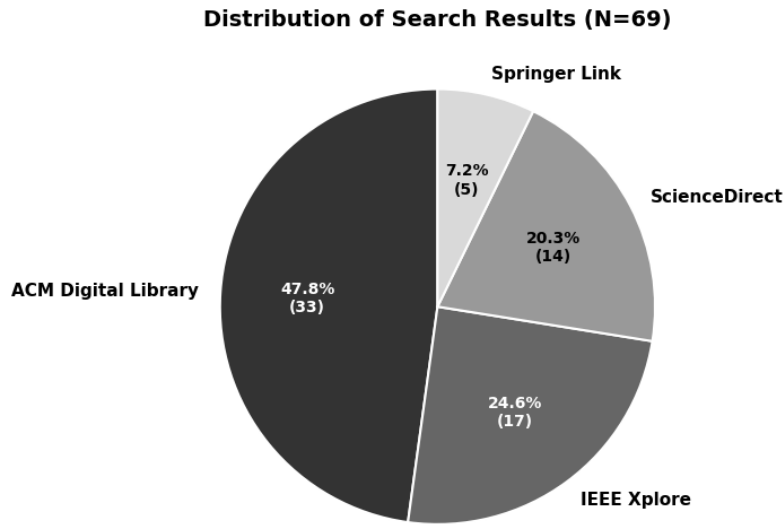


Figure A.1: Distribution of the 69 initial search results across the selected digital libraries.

To guarantee the results were focused on the research topic, the search was limited to the title, abstract, and keywords of the papers where possible.

Inclusion and Exclusion Criteria

To filter the initial set of results and select only the studies that directly address the research questions, a set of Inclusion Criteria (IC) and Exclusion Criteria (EC) were defined.

The **Inclusion Criteria** were as follows:

- IC1: The paper discusses or applies Large Language Models (LLMs) or transformer-based models.
- IC2: The paper must be written in English.
- IC3: The paper must have been published between 2020 and 2025.
- IC4: The study is peer-reviewed.

The **Exclusion Criteria** were defined to remove irrelevant studies:

- EC1: Papers not peer-reviewed (e.g., blogs, posters, editorials, or tutorials).
- EC2: Papers with insufficient technical detail.
- EC3: Papers written in languages other than English.

Figure A.2 summarises the study selection process.

The initial database search returned 69 records. After removing four duplicates, 65 records remained for screening. Of these, 37 were excluded after assessment against the inclusion and exclusion criteria, leaving 28 studies.

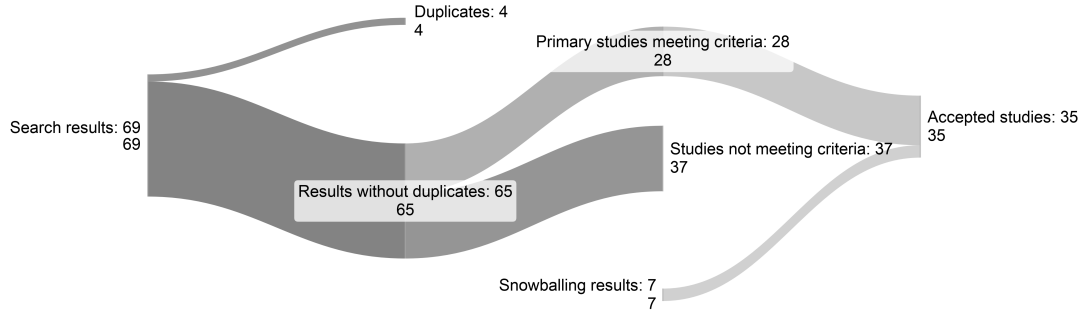


Figure A.2: Study selection flow from database searches and snowballing to the final sample.

Forward and backward snowballing added seven studies, resulting in a final set of 35 selected studies. These studies are identified as S01–S35 in Table A.2.

Table A.2: List of selected studies.

ID	Title	Authors	Year	Ref.
S01	LLM4Workflow: An LLM-based Automated Workflow Model Generation Tool	Xu et al.	2024	[41]
S02	AI for Low-Code for AI	Rao et al.	2024	[72]
S03	Auto-DSM: Using a Large Language Model to Generate a Design Structure Matrix	Koh	2024	[42]
S04	Automated Implementation Scheme for Low-Code Development Based on Large Language Models	Sun et al.	2024	[39]
S05	Automation in Model-Driven Engineering: A Look Back and Ahead	Burgueño et al.	2025	[62]
S06	ChatOps for Microservice Systems: A Low-Code Approach Using Service Composition and Large Language Models	Wang et al.	2024	[73]
S07	Complex Flowchart Text Generation Based on Large Language Models	Fang and Fan	2025	[74]
S08	Combining Low-Code Development with ChatGPT to Novel No-Code Approaches: A Focus-Group Study	Martins et al.	2023	[68]
S09	Enhancing Accessibility for Non-experts in Low-Code/No-Code Through the Addition of an AI Chatbot	Troyer et al.	2025	[75]
S10	D2: An LLM Agent Driving End-to-End Visual AI modeling in Energy Platforms	Li et al.	2025	[43]
S11	FlowPilot: A Generative AI-Driven Visual Language for Computational Thinking Education	Turchi et al.	2024	[40]
S12	Human-Centric Hybrid-AI for No-Code Development	Silva and Hildebrandt	2025	[48]
S13	Introducing LLMScript: A Turing Complete Prompt-Based Scripting Language for LLMs with No External Coding Required	Mowzoon and Faustini	2024	[76]
S14	Low-code solutions for business process dataflows: From modeling to execution	Eldin et al.	2025	[77]
S15	Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains	Cheng et al.	2024	[78]

Table A.2: List of selected studies (continued).

ID	Title	Authors	Year	Ref.
S16	Using Grammar Masking to Ensure Syntactic Validity in LLM-based modeling Tasks	Netz et al.	2024	[50]
S17	Turning Low-Code Development Platforms into True No-Code with LLMs	Hagel et al.	2024	[9]
S18	Neurosymbolic Repair for Low-Code Formula Languages	Bavishi et al.	2022	[46]
S19	InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs	Zhou et al.	2025	[47]
S20	White-box LLM-supported Low-code Engineering: A Vision and First Insights	Buchmann et al.	2024	[65]
S21	Automated Generation of AML Models for Industrial Plants Using LLM Chat Applications	Kunze and Fay	2024	[79]
S22	Power Grid Model Generation Based on the Tool-augmented Large Language Model	Deng et al.	2025	[80]
S23	Toward efficient vibe coding: An LLM-based agent for Low-Code software development	Malamas et al.	2025	[51]
S24	MetaCloze: a schema-guided automated building metadata model generation system via information extraction	He et al.	2025	[81]
S25	Constructing Enterprise Digital Twins by Augmenting LLMs with MDE	Barat et al.	2025	[82]
S26	Using (Not-so) Large Language Models to Generate Simulation Models in a Formal DSL: A Study on Reaction Networks	Kreikemeyer et al.	2025	[83]
S27	ChainBuddy: An AI-Assisted Agent System for Generating LLM Pipelines	Zhang and Arawjo	2025	[84]
S28	Generating SysML Behavior Models via Large Language Models: an Empirical Study	Wang et al.	2025	[38]
S29	Fully Autonomous Programming with Large Language Models.	Liventsev et al.	2023	[44]
S30	Program Synthesis with LLM-Predicted Minimal Specialized Grammars	Zarb	2025	[85]
S31	Fully Autonomous Programming Using Iterative Multi-Agent Debugging with Large Language Models	Grishina et al.	2025	[45]
S32	Exploring Influence Factors on LLM Suitability for No-Code Development of End User Applications.	Wang et al.	2025	[66]
S33	Software Model Evolution with Large Language Models: Experiments on Simulated, Public and Industrial Datasets	Tinnes et al.	2025	[10]
S34	“What It Wants Me To Say”: Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models	Liu et al.	2023	[20]
S35	Accurate and Consistent Graph Model Generation from Text with Large Language Models	Chen et al.	2025	[49]

For the temporal analysis, publication years were harmonised using the earliest publication year documented in the publisher’s records, including advance online publication. Bibliographic references retain the year of the cited edition. S04 is counted in 2024 based on the proceedings date, although its bibliographic reference is dated 2025. S09, S14, and S32 are counted in 2025 based on their online publication, although the cited editions are dated 2026.

A.3.3 Data Extraction and Synthesis

To ensure that we extracted and recorded the data systematically from each accepted study, we created an extraction form (Table A.3). The extraction form consists of two categories of fields. The first category includes fields that deal with the general information about each paper (i.e., Year Published, Study Type, Platform Characteristics), which allows us to see what areas of research are currently being pursued. The second category consists of fields that contain each study's contribution and enable us to answer our research question.

The fields in the first category include metadata related to each study, and therefore have a more structured format. This category's fields regarding Authors, Year, Source Venue allow us to understand where the current state of research is at concerning LLMs in Low-Code/MDE. The fields such as *Platform Type* and *License/Accessibility*, allow us to understand the openness of the platforms being researched and who they are intended to reach.

The second category includes fields tied to the research questions for this dissertation. For RQ1 (e.g., *LLM/Model Used*, *Model Access Type*, *Output Type*), we used fields to find architectural patterns and ways of integrating models. To address RQ2 regarding intermediate representations, we recorded data on *IR Type*, *IR Specification Method*, and *Persistence Format*. The fields such as *Syntax Enforcement Mechanism*, *Human-in-the-loop Level*, and *Validation Type* were used to address RQ3, which focuses on the correctness and reliability of the systems. Finally, the fields such as *Evaluation Method*, *Main Findings*, and *Reported Challenges* were used to address RQ4, and provided information on the *level of maturity and limitations of current solutions*.

Each study's extraction form was completed based upon the information available in each study. Some studies did not address one or more of the fields in the second category related to the research questions. When this occurred, the respective field was completed as "n/a" (not applicable) or "Not Reported". The fields in the second category that were qualitative (e.g., **Main Findings**, **Reported Challenges**) were completed with direct information from the studies. Since the fields were open-ended, and to provide a consistent interpretation of the results for the discussion in Section A.5, they were also summarised and grouped into common topics.

A.4 Results and Analysis

As outlined in Section A.3.3, we extracted both qualitative and quantitative data from the 35 selected sources detailed in Section A.3.2 by employing the data extraction forms. The purpose of this chapter is to report on the results of the qualitative and quantitative data extraction from the viewpoint of the research questions identified earlier. By doing this, it will synthesise the results to give an overall account of the current state-of-the-art of utilising LLMs in Low-Code and model-driven engineering systems.

Table A.3: Data extraction form fields.

Description	Type	Possible values (examples)	Associated RQs
<i>First group: General information</i>			
Title / Authors / Year	String Field	n/a	n/a
Source / Venue	Select One	ACM, IEEE, ScienceDirect, Springer, Other	n/a
Type of Study	Select One	Conference Paper, Journal, Workshop, Thesis	n/a
Platform Type	Select One	Low-code, No-code, Hybrid	n/a
License / Accessibility	Select One	Open Source, Commercial, Hybrid	n/a
<i>Second group: Contributions</i>			
LLM/Model Used	Select One	GPT, Llama, Claude, PaLM, Not used	RQ1
Model Access Type	Select One	API, Fine-tuned Local, Tool-Augmented	RQ1
Output Type	Select Many	Code, Model, Workflow, Rule	RQ1
Uses Intermediate Rep (IR) ?	Select One	Yes, No	RQ2
IR Type	Select Many	DSL, Graph, JSON, UML/Diagram, Grammar	RQ2
IR Specification Method	Select One	Grammar, Metamodel, No Specification	RQ2
Syntax Enforcement Mechanism	Select Many	Grammar Masking, Constrained Decoding, Parser-based	RQ3
Human-in-the-loop Level	Select One	Co-editing, Review-only, Iterative, Prompting	RQ3
Validation Type	Select One	Syntactic, Semantic, User Study, Formal	RQ3
Evaluation Method	Select One	Benchmark, Case Study, Experiment, Simulation	RQ4
Main Findings	String Field	n/a	RQ4
Reported Challenges	String Field	n/a	RQ4

The results are organised by research question. Section [A.4.2](#) presents integration approaches (RQ1). Section [A.4.3](#) examines intermediate representations (RQ2). Section [A.4.4](#) discusses correctness and validation mechanisms (RQ3). Section [A.4.5](#) summarises evaluation methods, benefits, and limitations (RQ4).

A.4.1 Study Metadata and Research Trends

The purpose of this section is to perform the quantitative analysis of the 35 selected studies included in the review. An overview of the temporal distribution, type of publication outlets, geographic location of the authors, and type of institution involved in the studies

will help us to gain a better understanding of the current state of the art in the area of LLMs applied to Low-Code development.

Figure A.3 shows the distribution of the 35 selected studies by year: one in 2022, three in 2023, twelve in 2024, and nineteen in 2025. Most selected studies were therefore published in the final two years of the review period.

The selected studies comprise 21 conference or workshop papers and 14 journal articles, as shown in Figure A.4. Conference and workshop papers are grouped together because they are published in proceedings. Figure A.5 shows the distribution of these publication types by year.

Figure A.6 groups the 35 studies by their authors' affiliations: 20 (57.1%) involve only universities or public/non-profit research institutions, 3 (8.6%) involve only companies, and 12 (34.3%) include both types. The mixed category also includes studies in which an author lists both an academic/research affiliation and a company affiliation.

Figure A.7 shows study counts by country, while Figure A.8 shows author counts based on the affiliations listed in each paper. Each author is counted once per paper and country, so authors may contribute to more than one country and are counted again when they appear in another paper. China has the highest author count (39), followed by the United States (32) and Germany (21). Hong Kong affiliations are included under China, while Taiwan is shown separately.

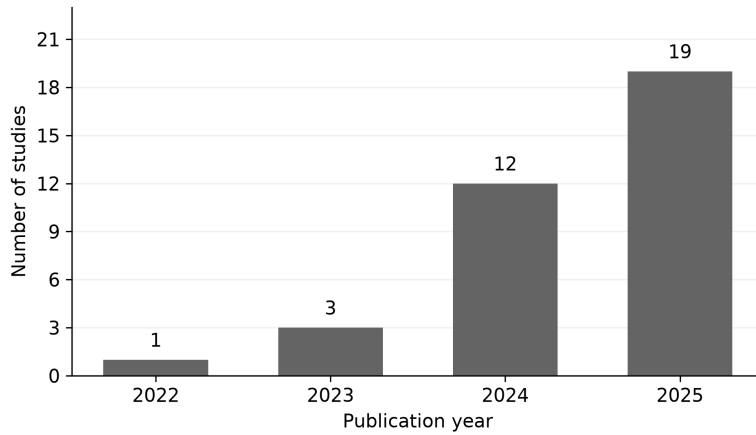


Figure A.3: Distribution of selected studies per year (2022–2025).

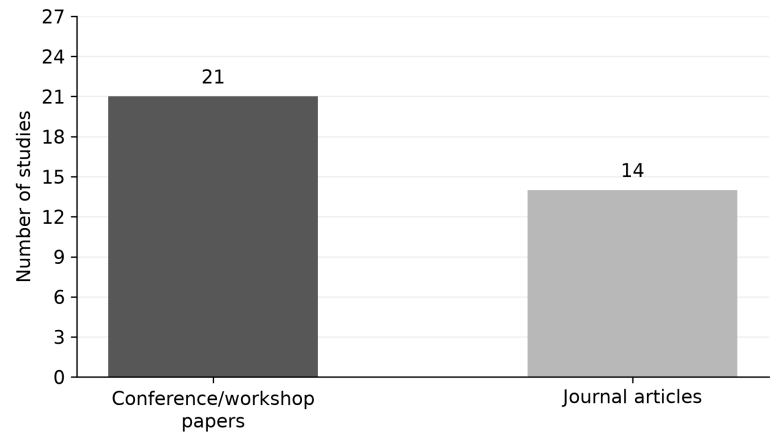


Figure A.4: Distribution of selected studies by publication venue type.

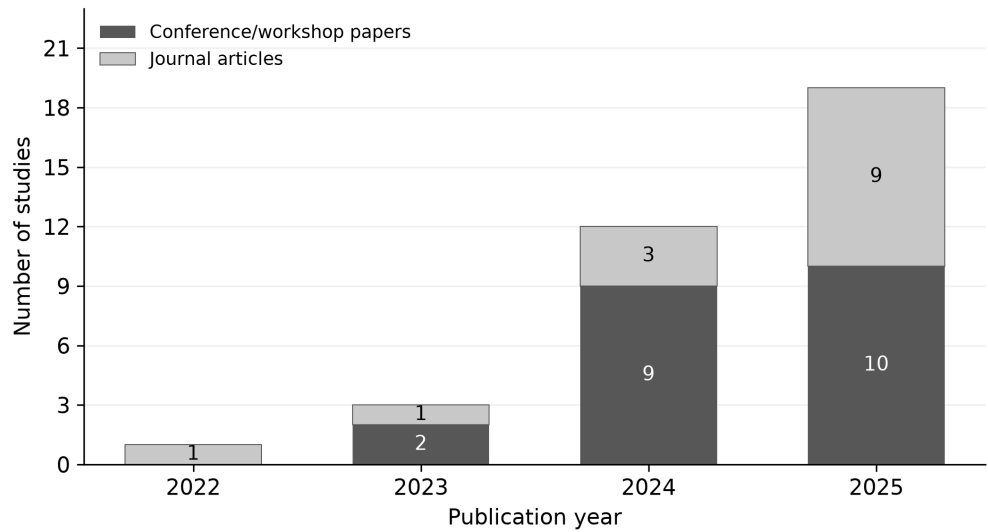


Figure A.5: Evolution of publication venues over the years.

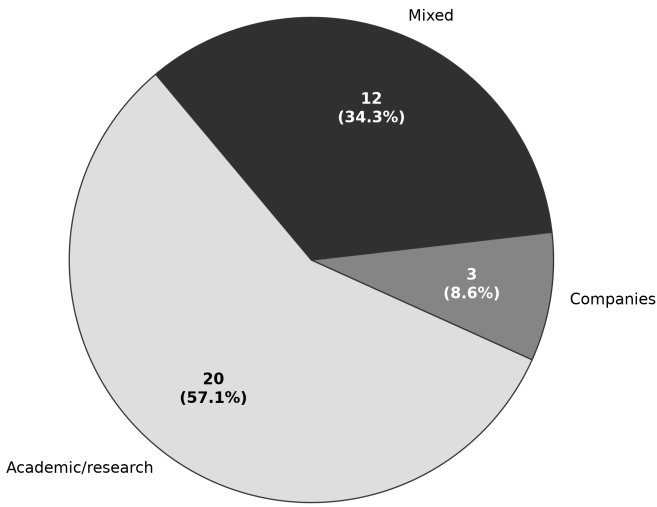


Figure A.6: Distribution of studies by author affiliation type.

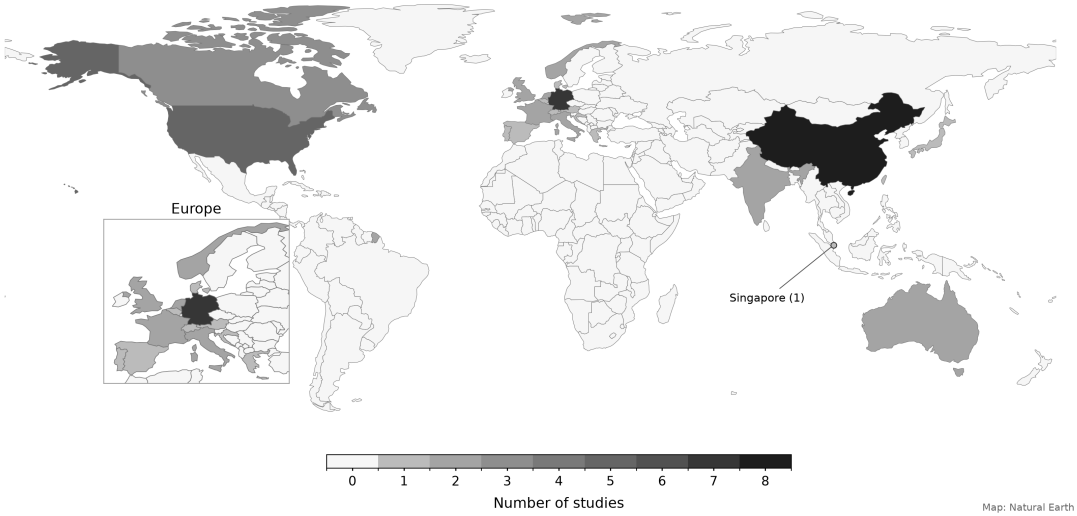


Figure A.7: Geographical distribution of selected studies on a world map.

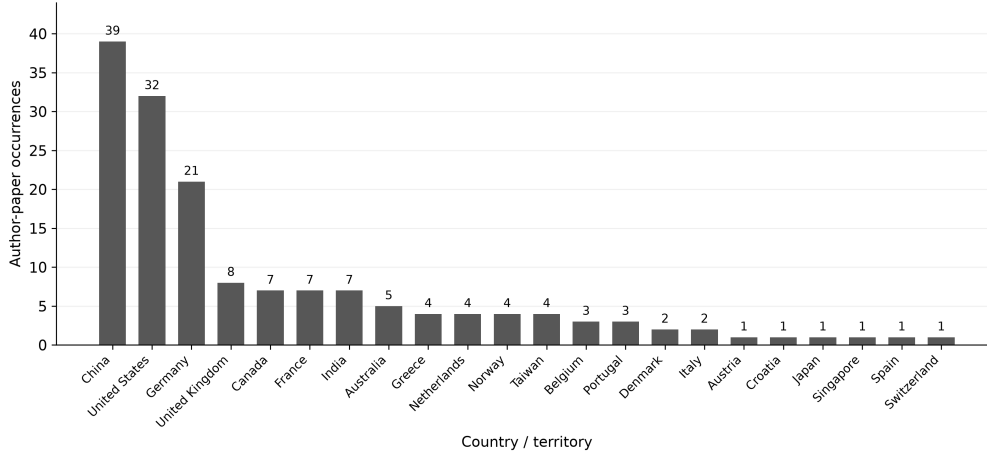


Figure A.8: Number of authors per country, based on their institutions.

A.4.2 RQ1: How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?

The subject of RQ1 is how LLMs are being incorporated into Low-Code Model-Driven Systems. According to the literature, there was no single methodology for this; rather, there were many methodologies with varying degrees of technology and autonomy within the systems. Table A.4 shows the *integration methods* identified.

An evaluation of the selected studies shows that the integration of LLMs into modelling environments has evolved past simple interaction interfaces. As indicated in the literature, there appears to be an evolution in the architectural control flow from simple interaction interfaces to autonomous loops. Three types of integration levels were determined based on the degree of autonomy allowed to the model: stateless direct generation, retrieval augmented generation, and agentic feedback loops.

Stateless prompt engineering is one of the integration approaches identified in the review. Studies included in the review (i.e., [39], [38]) illustrate direct generation: a user enters a natural language description of their desired artefact, and the LLM generates the artefact in a single pass, without showing how it was generated. For example, Wang et al. [38] illustrate the application of stateless prompt engineering for generating SysML models directly from text. In a stateless interaction, the model does not retain information from previous requests unless it is included again in the input. The prompt can still contain examples, documentation, and project-specific constraints. These results typically require manual validation by the user.

RAG (retrieval augmented generation) is used in various research studies to provide additional project-specific context. The RAG strategy gives additional context (documentation, pre-existing code bases, metamodels) to the original input for the model to improve the quality of the produced outputs. Examples of how researchers have utilised RAG can be seen in [10] and [41]. Tinnes et al. [10] utilise the version history of a particular

project to serve as a knowledge base allowing the model to learn industry specific pattern recognition not available to models without this knowledge. Xu et al. [41] implemented a "Rewrite-Retrieve-Read" method to check if the correct API was retrieved before executing the generation phase thereby reducing the amount of error that will result from the specific domain.

Other reviewed approaches use agentic feedback loops or multi-agent systems. Different from usual linear pipeline architectures, these paradigms create a recursive environment where the system evaluates its own output. Liventsev et al. [44] propose a "Synthesise-Execute-Debug" loop where the system executes the generated code and uses the result of the compilation errors as input to the model to enable self-correction. Likewise, Grishina et al. [45] studied a system that generates programs, runs tests, uses failures to suggest fixes, and keeps the best programs for another round.

The selected studies describe both direct generation and approaches that combine generation with additional context, constraints, or feedback. Their results do not establish that one architecture is more effective across all tasks or that the generated artefacts are ready for production use without further validation.

Table A.4: Integration Techniques of LLMs in Low-Code Platforms

Technique	Studies
Prompt Engineering	[S04], [S06], [S07], [S08], [S09], [S11], [S13], [S14], [S17], [S20], [S21], [S25], [S28], [S30], [S32], [S34]
Multi-Agent Systems	[S01], [S10], [S15], [S19], [S22], [S23], [S27], [S29], [S31]
Grammar-Guided	[S12], [S16], [S18], [S26], [S30]
RAG (Retrieval-Augmented)	[S01], [S03], [S10], [S13], [S23], [S33]
Model Fine-Tuning	[S02], [S24], [S26], [S35]

The interaction paradigms employed define the usability of the platform (Table A.5). The Conversational (Chat) approach is common (13 studies), functioning as an assistant (i.e., [73]) that operates alongside the user.

However, in engineering contexts, the Human-Validated Generation (Human-in-the-loop) approach is critical (13 studies). Study [82] provides a good example of this paradigm, where the system produces a draft version of the model, which the engineer must validate before it is accepted.

On the other hand, Fully Automated approaches (8 studies), as exemplified in [49], attempt to create complete solutions without human intervention, which raises additional issues related to verifying consistency.

We evaluated the types of artefacts created as a result of using different integration models (Table A.6). The results indicated a variety of artefacts created. Code and logic

Table A.5: Interaction Paradigms between User and LLM

Paradigm	Studies
Conversational (Chat)	[S02], [S04], [S06], [S09], [S10], [S13], [S15], [S17], [S21], [S22], [S23], [S27], [S34]
Human-Validated (Draft & Approve)	[S01], [S08], [S11], [S12], [S14], [S18], [S19], [S20], [S25], [S26], [S28], [S32], [S33]
Fully Automated	[S03], [S07], [S16], [S24], [S29], [S30], [S31], [S35]

were identified in 10 studies, with an example being [40], where the model lets the user define a new block in the visual language by describing the behaviour they want. The model will generate the code in JavaScript for that block, and the user can connect it visually to other blocks. We also have executable system models (10 studies) with [79] as an important example, where the purpose is to automatically generate the topology of an industrial system in AutomationML (AML) by reading legacy physical documentation from brownfield plants.

The workflow area (8 studies) shows the ability to use LLMs to create process orchestrations. In [77] for example, this study proposes BPDML (Business Process Data Modeling Language), an extension of BPMN/DF-BPMN. The language introduces the concept of "Activity DataFlow", where each activity explicitly specifies inputs, outputs, data stores, and data operations. Lastly, conceptual visual models/graphs were identified in 5 studies. Study [49] addresses the generation of consistent graph models.

Table A.6: Types of Artefacts Generated by the Integration

Artefact Category	Studies
Conceptual Visual Models	[S03], [S10], [S28], [S33], [S35]
Executable System Models	[S01], [S12], [S16], [S17], [S21], [S22], [S23], [S24], [S25], [S26]
Workflow and Orchestration	[S01], [S06], [S07], [S14], [S15], [S19], [S27], [S32]
Code and Logic	[S02], [S04], [S11], [S13], [S18], [S20], [S29], [S30], [S31], [S34]

In conclusion, the integration of LLMs has evolved from simple text prompts to hybrid architectures restricted by grammars [50]. It is becoming increasingly clear that the generation of models for critical applications should always be structured, and validated by a human [48].

A.4.3 RQ2: What types of intermediate or structured representations are used?

The second Research Question was to determine what types of representations were used to bridge the natural language input and the final modelling artefacts. This question aimed to understand how the representation would facilitate the validation of the input and make it easier to visualise the structure of the input. To answer the second research question, the results were categorised into two groups: Language-Based Representations (Table A.7) and Data-Structure Based Representations (Table A.8).

Language-Based Representations are based on a formal textual syntax to define the models. As can be seen in Table A.7, the most frequent type of representation is **Domain Specific Languages** (DSLs), which are designed for a specific context, allowing for more concise code compared to general languages. A very good example is [48], where the proposal follows the premise that the LLMs alone are not trustworthy for critical systems, because they are non-deterministic and hallucinate. The solution is a hybrid approach, using the LLM just to translate the user's natural language to a DSL legible by humans, allowing traditional formal methods to handle the actual execution safely.

To verify that the generated language is valid, many studies utilise Grammars and Schemas (3 studies). [85] is the principal study that uses Backus-Naur Form (BNF) as a key intermediate representation. In this study, the LLM receives examples of problems and their respective "mini grammar" (BNF) and generates a new BNF grammar for the target problem. This reduces the search space and thus lets the algorithm find a solution significantly faster.

The standard modelling languages, such as Unified Modeling Language (UML) and SysML, are commonly utilised (3 studies), as demonstrated in [38]. This study employs PlantUML as a textual intermediate representation to generate SysML models. By using this format, the LLM can describe the behavioural semantics of complex systems, such as states and transitions, which are then rendered into the final visual diagrams.

At last, Controlled Natural Language (3 studies), provides an interface between the users and their code. Rather than presenting users with raw code [20], it presents "grounded utterances", structured natural language that explains the fundamental Python operations. Users can then see the intent (i.e., "Step 1: Create column 'Year' from 'Date'") before the actual execution of the code. Grounded utterances can help users understand the planned operations and check whether they match their requirements.

On the other hand, Data-Structure Based Representations are typically used when a user needs to execute their model directly through machine-based processing, or if they want to integrate it onto web based platforms (as listed in Table A.8). Among Data-Structure Based Representations, JSON and Structured Commands are the most common representations (9 studies). For example, [80] uses JSON object as an intermediate representation of the electrical model that it wants to represent.

Also, unlike linear code, Serialised Graphs and Diffs are used to represent graphs,

Table A.7: Formal and Language-Based Representations (Language-Based)

Representation Type	Studies
Domain-Specific Languages (DSLs)	[S06], [S12], [S14], [S16], [S17], [S23], [S26]
Modelling Languages (UML/SysML/AML)	[S21], [S28], [S35]
Controlled Natural Language	[S13], [S19], [S34]
Grammars & Schemas (BNF/EBNF/Ontology)	[S16], [S24], [S30]

as we can see in [49]. These formats include both the nodes and edges to preserve the topological integrity of the graph.

Table A.8: Structural and Data Representations (Data-Structure Based)

Representation Type	Studies
JSON and Structured Commands	[S01], [S02], [S04], [S10], [S15], [S19], [S22], [S27], [S32]
Serialised Graphs and Diffs	[S01], [S07], [S12], [S33], [S35]
Matrices	[S03]
Abstract Syntax Tree (AST)	[S18], [S20]

A.4.4 RQ3: What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?

The third Research Question investigates how the quality of the created models can be ensured. The studies have employed several methods for ensuring that the generated output is correct (Table A.9).

The most common method is Automated Verification and Repair, which is used in 8 studies. This method verifies the correctness of the LLMs generated artefacts by testing them. When they contain errors, such as incorrect results due to hallucination, it attempts to repair them automatically. A notable study using an automated verification and repair methodology was [43]. This study uses a "Validation by Execution" technique. A predictive workflow is generated and tested for accuracy. If the model performance is low, the agent enters an optimisation loop, automatically suggesting repairs, from adjusting parameters to modifying features, until the results meet the required standard.

Three studies used constraint-based generation. In Netz et al. [50], grammar masking restricts generation to outputs that follow the specified grammar. This prevents the syntactic errors covered by that grammar, but does not establish semantic correctness or remove the need to check whether the generated model represents the intended requirements.

Static Analysis and Filtering (5 studies) are used for validating the generated output (rules, syntax, grammar) without running the model in a real-world environment. The

studies [9] and [51] can be seen as examples of how this method is applied. In Hagel et al. [9], as soon as the DSL code is generated, it is validated directly by the platform’s compiler to catch syntax errors. Similarly, [51] focused on achieving the maximum "Valid Model Rate" by validating that the output meets the DSL grammar, thus ensuring that it could run.

Tool-assisted human verification is reported in seven studies. It allows users to inspect generated artefacts and request corrections. This provides an opportunity to identify errors, but does not guarantee that every error will be detected or corrected.

Table A.9: Correctness Assurance Mechanisms

Strategy / Mechanism	Studies
Automated Verification and Repair	[S10], [S18], [S22], [S28], [S29], [S31], [S32], [S35]
Tool-Assisted Human Verification	[S01], [S02], [S04], [S07], [S12], [S33], [S34]
Static Analysis and Filtering	[S02], [S14], [S17], [S23], [S24]
Constraint-Based Generation	[S16], [S26], [S30]

A.4.5 RQ4: How are these approaches evaluated, and what benefits or limitations are reported?

This final research question aims to complement the technical discussion of the integration methods by looking at the practical impact of LLMs in Low-Code development. It examines the evaluation methodologies used and the reported advantages and limitations, as seen in Tables A.10 and A.11, respectively.

The selected studies report both benefits and limitations. Some report improvements in efficiency for the tasks evaluated, while others identify semantic errors and the effort required to review or correct generated artefacts. Because the studies use different tasks and evaluation methods, their results do not establish a general productivity advantage.

Reported efficiency gains need to be interpreted in the context of each evaluation. One of the ways improvements in efficiency were evaluated was through the use of a formula proposed by Sun et al. [39] for measuring improvement, where it is considered *Time*, *Cost*, *Errors* as main factors: $E = (W_t \times \Delta\mathbf{Time}) + (W_c \times \Delta\mathbf{Cost}) + (W_e \times \Delta\mathbf{Errors})$.

Using this expression applied to data from JD.com/China Unicom, they reported a 60% increase in Efficiency (the **E** in the formula) in contrast to the traditional ways of developing. Likewise, comparative studies on accessibility by Hagel et al. [9] and Malamas et al. [51], measured "Valid Model Rate" and "time-on-task", both found that the use of AI-based tools has greatly lowered the threshold of entry for non-expert developers as well as reduced the amount of time required for developing first-draft DSL code.

However, the increased productivity can be negatively impacted by the lack of quality and reliability concerning semantic consistency. Although grammar-guided decoding has

greatly reduced the number of syntactic errors (as shown in [50]) the semantic hallucination continues to be a major problem. Both Wang et al. [38] and Tinnes et al. [10] reported that the model generates syntactically correct artefacts, but does not capture the project-specific logic or industry constraints unless provided with external context (RAG). Reviewing and debugging the generated artefact takes additional time and can reduce the time saved during generation.

Liventsev et al. [44] evaluated a Synthesize-Execute-Debug approach that tests generated programs and uses feedback from failed tests to guide repair. Using OpenAI Codex on Program Synthesis Benchmark 2, their framework outperformed generation without a repair phase and the genetic programming approaches used for comparison.

Ultimately, the evaluation results suggest that although LLMs provide a demonstrable increase in both speed and accessibility, there is a need for strong verification methods to ensure that LLMs will be viable in industry. It appears that the transition from "generation", to "iterative repair" is the key to transforming unreliable probabilistic output into reliable engineering models.

Table A.10: Evaluation Methodologies

Evaluation Methodology	Studies
Automated Benchmarks and Datasets	[S07], [S16], [S18], [S24], [S26], [S28], [S29], [S30], [S31], [S35]
Industrial Case Studies & Pilots	[S01], [S03], [S04], [S10], [S11], [S13], [S14], [S20], [S21], [S22], [S25], [S33]
Controlled User Study	[S02], [S06], [S09], [S15], [S17], [S19], [S23], [S27], [S32], [S34]
Focus Groups, Surveys & Vision	[S05], [S08], [S12]

The literature has identified a trade-off between the efficiency and reliability in using LLMs as described in Table A.11. As stated earlier, productivity was found to be the most common positive outcome when using LLMs for development, because they automate many repetitive tasks associated with code generation and configuring workflows. However, reviewing and correcting generated output also takes time and must be considered when assessing productivity.

Although Natural Language Processing (NLP) has greatly enhanced the Accessibility and Usability of software engineering (making it easier for many people to develop applications) as well as the participation of non-technical users in the application development process through enabling them to describe the application's requirements using natural language (as seen in [68] and [20]), there are still usability barriers that exist.

For example, vague user prompts produce poor results and because most users are unaware of the abstraction gap between the user's intent and the machine-readable form of the user input [[72], [51]], they may have difficulty obtaining useful results from an NLP based tool.

The reviewed studies also report limitations in the quality and reliability of generated artefacts, despite the use of automated checks and repair [[44], [45]]. The non-deterministic nature of LLMs is the primary reason for hallucinations, which can create considerable threats to high-stakes applications. As seen in [38], syntactic errors were mostly resolved, but semantic issues persist, suggesting that the feedback mechanism using formal verification techniques needs further investigation to help LLMs better understand complex requirements and logical relationships.

Table A.11: Reported Benefits and Limitations

Impact Dimension	Benefits (+)	Limitations (-)
Efficiency and Productivity	[S03], [S04], [S10], [S14], [S15], [S17], [S19], [S25], [S32], [S33]	[S16], [S27]
Accessibility and Usability	[S02], [S08], [S09], [S11], [S20], [S23], [S34]	[S02], [S11], [S19], [S27]
Quality and Reliability	[S29], [S31], [S33], [S35]	[S07], [S13], [S21], [S26], [S27], [S28]

A.5 Discussion

While we managed to acquire some insights regarding our research questions thanks to the selected studies, it is pertinent to look at what these findings mean for the LLM use in Low-Code and model-driven platforms.

A.5.1 Common Architectures for Integrating AI

We verified that the integration of Large Language Models (LLMs) into Low-Code platforms has evolved over time in terms of their degree of complexity and how their integration technique, ranging from simple external assistants to deep, contextualised integration.

One approach is to integrate the LLM as a copilot or side-by-side assistant. This type of architecture was used by [40], for example, in which the LLM is positioned as a separate service or a ChatBot or code completion tool layered over the existing IDE. The benefits of using this type of architecture are numerous, including being easy to implement since no modifications to the underlying platform are needed. However, the main drawback of this type of architecture is that the LLM is unaware of the larger project state. [10] states that, with a lack of understanding of the larger project state or the modelling context, the LLM fails to predict correct evolutions, generating artefacts that may be syntactically correct but are functionally incorrect relative to the rest of the application.

Because of this, a significant number of researchers recommend using retrieval-augmented generation (RAG) architecture to address the drawbacks associated with the copilot architecture. RAG architectures align with much of the more extensive AI

literature Lewis et al. [86] that uses context to provide additional information to the model prior to generating output. As highlighted in [10] that the LLM needs to understand the environment and context it is in, so this study found that the use of RAG architecture significantly improves the performance of industrial applications of LLM. RAG can provide project-specific documentation and examples before generation. The generated artefacts still require validation against the target language and platform constraints.

Some reviewed approaches organise generation, testing, and repair into an iterative process. Liventsev et al. [44] and Grishina et al. [45] use test results to guide changes to generated programs. These approaches allow candidate solutions to be revised after execution, rather than relying on a single generation step.

Liventsev et al. [44] found that combining code repair with the generation of new alternatives gave better results than using either strategy alone in their experiments. The reviewed studies also indicate a trend toward hybrid neuro-symbolic architectures in which the probabilistic output of the LLM is constrained and validated by the deterministic output of the Low-Code engine.

A.5.2 The Importance of Intermediate Formats

Traditional Model-Driven Engineering uses extremely complex file formats, such as XML Metadata Interchange (XMI) and large JSON schema definitions. The issue with these large and rigidly-structured files is that they are difficult for LLMs to process correctly, their complex formatting makes it harder for the model to understand the structure. Similarly, trying to have LLMs generate the final file using these formats, generally results in failures. An effective solution to this problem is to define an (IR), which will act as a bridge between the LLM and the actual final system that is being created, and it helps to understand what the model actually generates

An important realisation regarding IRs is that they should also be readable by humans. This requirement aligns with many of the more recent principles of Human-Centred AI [87]. Specifically, one of the primary principles of Human-Centred AI is that to achieve both safety and trust, AI systems must provide users with high levels of interpretability and controllability. However, if the AI generates a complex block of code, then, the user will not be able to verify whether or not the code is correct. Therefore, to overcome this issue, researchers such as Liu et al. [20] proposed using "Grounded Utterances," i.e., structured text-based steps that describe the logical flow of the code in plain English. Thus, the user can quickly determine the intent behind the AI-generated code and evaluate whether or not the code is correct by looking at it, reducing significantly the cognitive barrier between vague and informal requirements and formal syntax. Along with this, researchers such as Silva and Hildebrandt [48] proposed developing a DSL that is easier for the average user to read and understand than complex graphs, thus providing more clarity into how the system behaves from the perspective of the domain expert.

In addition, using simpler intermediate formats reduces the risk that the AI will become

confused about the structure of the data, consistent with Amershi et al. [88] guidelines for minimising ambiguity when interacting with humans. Additionally, researchers such as Zhou et al. [47] demonstrated that having the model generate "pseudocode" prior to generating a visual graph (or other output) is far more likely to produce reliable results than attempting to generate a visual graph directly from the model, due to the reduced amount of syntactical errors present in larger JSON files.

Ultimately, while the selection of an appropriate intermediate representation is a technical decision, it also means an important non-technical decision, since the selected IR must be simple enough for the LLM to process, but also clear enough for the human user to trust.

A.5.3 Ensuring Reliability in AI-Generated Models

Intermediate formats represent a step towards improving the interpretability of Generative Models, however, they do not overcome the fundamental probabilistic nature of generative models. Unlike the usual, deterministic tools, LLMs will produce plausible, although incorrect outputs, an issue famously described by Bender et al. [89] as the "Stochastic Parrots" problem. The review has demonstrated that the reliability of generative models represents a multi-layered challenge that evolves from simple syntax validation to complex hybrid validation methods.

One of the important distinctions found within the literature is between syntactic validity (is the code valid? Does the code compile?) and semantic correctness (does the code perform the function requested by the user?). Netz et al. [50] demonstrate that techniques such as "Grammar Masking" can be used to enforce the strictest adherence to a language's syntax, resulting in near perfect syntactic validity. However, as Wang et al. [38] point out, a model may generate syntactically correct code that still does not perform as expected due to hallucinations, where the model produces incorrect logic despite valid syntax. [44] refer to this as "Near Miss Syndrome", where the output appears correct upon initial inspection, however, upon execution it performs incorrectly.

Some reviewed approaches use iterative repair to address semantic errors. Liventsev et al. [44] use feedback from failed tests to guide the repair of generated programs. Grishina et al. [45] generate candidate programs, test them, use the detected failures to guide repair, and rank the candidates for further iteration.

The reviewed literature also explores combining neural models with symbolic methods to support reliability. Silva and Hildebrandt [48] propose connecting generative AI and symbolic models through a formal DSL that users can understand. Bavishi et al. [46] use neural techniques to locate errors and rank possible repairs, while symbolic methods generate repairs for low-code formulas.

A.5.4 Balancing Speed with the Need for Review

Although LLMs are capable of reducing the physical effort involved in coding, the gain in productivity is largely counterbalanced by the cognitive cost of verification. Zhou et al., [47] found that most of the time, users "pause" and evaluate whether the AI's result was intended to achieve what the user originally wanted to do, thus turning the user into an "auditor" rather than an author. The fact that users need to validate the results also complicates the evaluation process because, as Tinnes et al., [10] point out, although the generated model may be syntactically correct, it may still fail when executed in the context of the current project. In the end, the effectiveness of AI-assisted platforms is based upon how quickly and efficiently they can reduce the effort needed by humans to verify their work.

Productivity gains should therefore be assessed using the total time required for generation, review, and correction. Faster generation does not necessarily reduce the overall effort if extensive correction is still needed.

A.5.5 Current Trends and Industry Interest

The main aspect of the reviewed literature is the shift from theoretical research on integrating Large Language Models (LLMs) into practical, industrial research applications. Some of the reviewed studies were conducted in collaboration with industrial partners and originated from leading industrial firms, suggesting a shared commitment among major market participants to invest in this area and democratise software modelling.

In addition to the "Big Tech" players, significant interest exists in specialised engineering areas. Examples include the development of Enterprise Digital Twins using metamodel-augmented LLMs (to ensure that generated simulations really show complex constraints inherent in an organisation) by researchers working at TCS Research [82].

Industry investment is pushing the trend toward greater autonomy. While earlier versions of such systems operated as passive copilot systems, the latest version of the research being done (e.g., ChainBuddy [84] and the iterative loop-based study done by [44]), suggests that future versions of such systems will operate as semi-autonomous. These semi-autonomous agents will not only be able to generate code, but also to plan, execute, and correct workflow sequences independently of human operation, therefore, human oversight of their operation will be necessary.

Silva and Hildebrandt [48] propose a research agenda for no-code development that combines generative AI with symbolic models. Their approach uses an intermediate domain-specific language to describe behavioural scenarios in a formal representation designed for users to read and understand.

The reviewed examples combine LLM generation with structured representations, execution feedback, or validation [44, 48, 82, 84]. They explore different architectures, and the review does not establish a single approach suitable for all industrial applications.

A.6 Threats to validity

In the next subsections we discuss the threats to validity of this work under several perspectives, following the classification of construct, internal, external, and conclusion validity.

A.6.1 Construct validity

Regarding the search string, we included "Low-Code", "Large Language Models" and "Model-driven Engineering". The terminology related to this subject is changing very quickly, so it is possible that there are some additional studies that we missed because they were using other names for these types of technology. In addition, we limited our searches to four major online libraries: ACM, IEEE, ScienceDirect and Springer. Since research in Artificial Intelligence moves very fast, we might have missed some important recent work that has not yet been published in traditional journals or conferences.

A.6.2 Internal validity

To ensure a rigorous process, we utilised Parsifal tool and worked within a well-defined protocol. As such, the decision to include or exclude articles as having insufficient technical detail may be somewhat subjective. We helped to mitigate some of these issues by performing a snowballing process to locate other relevant articles in the references cited in each article we located during our first search. However, we believe that the selected articles cover most of the high-quality publications in LLMs for Low-Code and model-driven platforms.

A.6.3 External validity

One major limitation is that the rate of technological advancement is so rapid that many of the articles we reviewed were published between 2024 and 2025. The studies evaluated specific models and versions, so their findings may not generalise to later models. Another limitation of prior research is that most of the studies have been conducted by researchers who work in universities, for that reason, they typically test prototypes rather than commercially available products. In addition, because university-based testing environments tend to be much less complicated than commercial environments, the results may be very different when applied to complex legacy systems used in a company setting.

A.6.4 Conclusion validity

Finally, this validity refers to the relationship between our conclusions and the extracted data. We used a specific form to extract data from every paper systematically. However, some fields, such as the "Reported Challenges", required us to interpret the text written by the original authors. Different researchers might interpret these descriptions differently. Additionally, the various papers used very different ways to evaluate their tools, ranging from user surveys to automated benchmarks. This variety makes it difficult to make a direct comparison and say for sure which method is the best. Ultimately, we expect more research in these areas to benefit both academia and the Low-Code industry, promoting the development of trustworthy and efficient AI-enhanced modelling platforms.

A.7 Conclusions and future work

We aimed to learn and understand more about how Large Language Models (LLMs) have been integrated into Low-Code and Model-Driven Engineering (MDE) systems. This review analysed 35 selected studies published between 2022 and 2025. We followed a defined protocol to identify the integration architectures, the intermediate representations used to structure the output, and the validation mechanisms required to ensure the correctness of the generated models.

The reviewed studies describe approaches ranging from direct prompting to pipelines with structured intermediate representations and validation. Intermediate representations, such as DSLs, JSON, XML, and graphs, connect natural-language input with the formats required by modelling platforms. They can also make generated artefacts easier to inspect and check before use.

Some studies report reductions in the time needed to create initial drafts and models. However, reviewing and correcting generated output also takes time and should be considered when assessing productivity. Human review, automated repair loops, and deterministic checks are among the approaches used to identify and address errors. Differences in tasks, tools, and evaluation methods limit direct comparison between these approaches.

For future work, we suggest three main directions based on the gaps we identified. Firstly, there is currently no standardisation regarding the benchmarks used to evaluate the relative qualities of various tools. Because each of the studies we reviewed utilised a unique methodology to assess success, it is difficult to compare the results of the individual studies. Establishing a common set of tasks that will be used to benchmark modelling capabilities will enable the community to identify which architectural approaches offer advantages over others. Second, we believe that future investigations should focus on developing "semantic validation" capabilities. Most existing tools are capable of identifying syntactical errors (i.e., grammar errors) in models, however, they continue to struggle with

identifying logical errors (i.e., semantic errors). Third, we believe that a greater emphasis should be placed upon developing intermediate representations that are both clearer and more intuitive to users, even if they do not have any coding experience. Developing a clear and easy to envision intermediate representation is important because it enables users to clearly visualise the output produced by the model, facilitating inspection and correction of the generated output. Ultimately, this helps to align the model's production of output with the user's intended outcome.

These findings informed the design of the DISME wizard presented in Chapters 4 and 5. The wizard combines LLM generation with structured representations, deterministic checks, and user review. These mechanisms support the detection and correction of errors before import, while the evaluation in Chapter 6 examines the limitations that remain in practice.

VALIDATION MATERIALS AND SESSION EVIDENCE

B.1 Artefact Quality Rubric

The rubric in Table B.1 was used to compare the initial and confirmed artefacts with the reference specification for the reduced conference scenario. Criteria marked with an associated critical failure identify differences that could prevent or change the intended workflow even when the total quality score remained high.

Table B.1: Artefact quality rubric for the reduced conference scenario.

ID	Phase	Criterion	Associated critical failure
R01	0	The domain represents a scientific conference and the main entity is <code>Article</code> .	The <code>Article</code> entity is missing.
R02	0	A workflow containing article submission followed by a decision is identified.	–
R03	1	There is one main <code>Article</code> entity, or a semantically equivalent entity.	The <code>Article</code> entity is missing or replaced by an incompatible concept.
R04	1	<code>Title</code> exists, uses a text type, and is mandatory.	A required property is missing.
R05	1	<code>Abstract</code> exists, uses a text type, and is mandatory.	A required property is missing.
R06	1	<code>File</code> exists, uses a file-compatible type, and is mandatory.	A required property is missing.
R07	1	<code>Status</code> exists, is mandatory and enumerated, and keeps only one current value.	<code>Status</code> is missing or allows several current values at the same time.
R08	1	<code>Status</code> supports <code>Submitted</code> , <code>Accepted</code> , and <code>Rejected</code> , or semantically equivalent values.	The available values prevent the expected submission or decision.
R09	1	<code>Submit Article</code> is identified as the transaction that creates an <code>Article</code> .	The <code>Article</code> is linked to an incorrect creation transaction.
R10	2	The <code>Author</code> and <code>Conference Chair</code> roles exist, or semantically equivalent roles are used.	One of the required roles is missing.
R11	2	A process covering article management, submission, and decision exists.	–
R12	2	<code>Submit Article</code> exists, or a semantically equivalent transaction is used.	The transaction is missing.
R13	2	The <code>Author</code> requests and executes <code>Submit Article</code> .	Responsibility is assigned to the wrong role.
R14	2	<code>Submit Article</code> starts the process and produces the <code>Article</code> .	The transaction does not create the <code>Article</code> or start the workflow.

Table B.1: Artefact quality rubric for the reduced conference scenario (continued).

ID	Phase	Criterion	Associated critical failure
R15	2	Record Decision exists, or a semantically equivalent transaction is used.	The transaction is missing.
R16	2	The Conference Chair executes Record Decision after submission.	Responsibility is assigned to the wrong role.
R17	2	Record Decision ends the process.	The decision does not allow the process to finish.
R18	3/3b	No unnecessary form query is created. Any added query has a reason and uses valid references and filters.	An invalid query is linked to the wrong records.
R19	3/3b	No unnecessary logic query is created. Any added query is necessary for the scenario and valid.	An invalid query changes the rule's behaviour.
R20	4	Submission presents a form containing Title, Abstract, and File.	A missing field prevents creation of the expected Article.
R21	4	The three submission fields are mandatory and use compatible types.	A required property cannot be entered.
R22	4	The submission rule creates the Article in the correct context.	The rule creates or changes the wrong record.
R23	4	Submission assigns only Submitted as the current Status.	The initial Status is incorrect or several current values are assigned.
R24	4	Submission presents a confirmation to the Author.	–
R25	4	After submission, Record Decision becomes available in the same workflow.	The decision cannot be started for the submitted Article.
R26	4	The decision acts on the Article that originated the transaction.	The rule reads or changes a different Article.
R27	4	The decision checks that the current Status is Submitted before continuing.	A decision is allowed when the Article has an incompatible Status.
R28	4	The Conference Chair selects exactly one value: Accepted or Rejected.	Both values are assigned, or neither can be selected.
R29	4	The selected value replaces the current Status and preserves the expected history.	The final Status does not match the selection or several current values remain.
R30	4	The decision presents a confirmation, ends the process, and both rules are structurally valid and can be opened in DISME.	A rule is structurally invalid or cannot be converted for import.

For R20, the score is 1 when all three required fields (Title, Abstract, and File) are present, 0.5 when one is missing, and 0 when two or more are missing. A missing field that prevents creation of the expected Article is also recorded as a critical error. Partial credit therefore does not necessarily mean that an error is non-critical.

B.2 Session-Level Results

Table B.2 summarises the main results retained for each of the 19 analysed sessions. P17 is not listed because it was excluded during data preparation, as described in Section 6.4.1. Quality values are rubric points out of 30 and are presented as initial/confirmed values. Critical error counts follow the same initial/confirmed order. For assistance, P, S, and T represent procedural, semantic, and technical assistance, respectively. Import values report structural models followed by Action Rule drafts. The complete functional

APPENDIX B. VALIDATION MATERIALS AND SESSION EVIDENCE

path required both transactions to finish for the same article and the expected current and historical states to be verified.

Table B.2: Task, quality, assistance, import, and execution results for the 19 analysed sessions.

Session	Tasks	Quality I/F	Assistance	Critical I/F	Import S/R	Functional path
P01	7/7	28.5/29.5	P, S	1/0	Yes/Yes	Yes
P02	7/7	28.5/29.5	P, S, T	1/0	Yes/Yes	Yes
P03	7/7	30/30	P, S, T	0/0	Yes/Yes	Yes
P04	7/7	29/29	S	0/0	Yes/Yes	Yes
P05	7/7	30/30	S	0/0	Yes/Yes	Yes
P06	7/7	29.5/29.5	P, S, T	0/0	Yes/Yes	Yes
P07	7/7	25/25	P, S	4/4	Yes/Yes	No
P08	7/7	29/29	S	1/1	Yes/Yes	No
P09	7/7	30/30	S	0/0	Yes/Yes	Yes
P10	7/7	29.5/29.5	S	0/0	Yes/Yes	Yes
P11	7/7	30/30	S	0/0	Yes/Yes	Yes
P12	7/7	30/30	P, S	0/0	Yes/Yes	Yes
P13	7/7	28/28	P, S	1/1	Yes/Yes	Yes
P14	7/7	30/30	P, S	0/0	Yes/Yes	Yes
P15	7/7	30/30	S	0/0	Yes/Yes	Yes
P16	7/7	30/30	S	0/0	Yes/Yes	Yes
P18	7/7	29/29	S	1/1	Yes/Yes	Yes
P19	7/7	28.5/28.5	P, S	2/2	Yes/Yes	Yes
P20	7/7	27.5/28	P, S	0/0	Yes/Yes	Yes

In P20, R11 received 0.5 because the process included submission and decision, but also an unnecessary classification transaction. R16 received 0.5 because the Conference Chair was responsible for the decision, but the sequence after submission was incorrect. In comparison, P18 received 0 for R16 because the decision was assigned to the Author instead of the Conference Chair. P20 received 0 for R25 because the submission rule initiated an additional classification transaction instead of establishing the intended causal link to Record Decision. Both submission and decision were nevertheless completed for the same Article. This was not a critical failure under R25 because the decision could still be started.

STUDY INSTRUMENTS

This appendix reproduces the instruments presented to participants. The Portuguese wording is retained because it was the wording used during the study. English translations are provided for readability and were not shown to participants.

C.1 NASA-TLX Questionnaire

NASA-TLX was completed after the common workflow and before free exploration. Participants were instructed to assess only their own use of the wizard during the common workflow, excluding familiarisation, free exploration, and the operations performed by the moderator in DISME. The questionnaire was presented through the HTML interface used in the study¹ and contained the six NASA-TLX dimensions [57].

The HTML returned one value for each dimension in increments of five from 5 to 100. For mental demand, physical demand, temporal demand, effort, and frustration, higher values represent a higher perceived level of the dimension. For performance, the HTML presents perceived success from low to high, so a higher value represents better perceived performance. This performance orientation differs from the original NASA-TLX rating sheet, which presents performance from good to poor. The values were retained in the orientation shown to participants and were analysed separately rather than combined into an overall score.

Table C.1: NASA-TLX items presented through the study HTML.

Dimension	Portuguese wording used	English translation	Direction
Mental demand	<i>Exigência mental: A nível mental, quão exigente foi a tarefa?</i>	How mentally demanding was the task?	Very low–very high
Physical demand	<i>Exigência física: A nível físico, quão exigente foi a tarefa?</i>	How physically demanding was the task?	Very low–very high
Temporal demand	<i>Exigência temporal: Quão apressado foi o ritmo da tarefa?</i>	How hurried was the pace of the task?	Very low–very high

¹<https://eelab.arditi.pt/nasa-tlx/nasa-tlx.html>

APPENDIX C. STUDY INSTRUMENTS

Table C.1: NASA-TLX items presented through the study HTML (continued).

Dimension	Portuguese wording used	English translation	Direction
Performance	<i>Performance: Quão bem-sucedido(a) foi ao realizar o que lhe pediram para fazer?</i>	How successful were you in completing what you were asked to do?	Very low–very high
Effort	<i>Esforço: Com que intensidade trabalhou para atingir o seu nível de desempenho?</i>	How hard did you work to achieve your level of performance?	Very low–very high
Frustration	<i>Frustração: Quão inseguro, desanimado, irritado, stressado e aborrecido se sentiu?</i>	How insecure, discouraged, irritated, stressed, and annoyed did you feel?	Very low–very high

C.2 Final Questionnaire

The final questionnaire was identified only by the participant code. Participants were informed that there were no correct answers and were asked to respond based on the session they had just completed. The questionnaire did not request names or email addresses.

C.2.1 Previous Experience

Table C.2: Participant profile questions.

Portuguese wording used	English translation	Response options
Antes desta sessão, qual era a sua familiaridade com processos de submissão e decisão de artigos em conferências científicas?	Before this session, how familiar were you with article submission and decision processes in scientific conferences?	Nenhuma; Pouca; Alguma; Muita
Qual é a sua experiência com ferramentas low-code ou de modelação?	What is your experience with low-code or modelling tools?	Nenhuma; Pouca; Alguma; Muita
Com que frequência utiliza ferramentas de IA generativa?	How often do you use generative AI tools?	Nunca; Menos de uma vez por mês; Mensalmente; Semanalmente; Diariamente
Antes desta sessão, qual era a sua experiência com o DISME?	Before this session, what was your experience with DISME?	Nenhuma; Pouca; Alguma; Muita

C.2.2 System Usability Scale

The questionnaire used a published European Portuguese version of the ten-item System Usability Scale [55, 56]. In this block, *produto* referred to the wizard as experienced during familiarisation, the common workflow, and free exploration. Participants were told to consider the wizard’s description, interview, review, and import steps, but to exclude the technical operations performed by the moderator in DISME and any limitations of the TeamViewer connection. Every item used a five-point scale from 1, *Discordo totalmente*, to 5, *Concordo totalmente*.

Table C.3: System Usability Scale items used in the final questionnaire.

Item	Portuguese wording used	English translation
SUS-01	Acho que gostaria de utilizar este produto com frequência.	I think that I would like to use this product frequently.
SUS-02	Considerei o produto mais complexo do que necessário.	I found the product more complex than necessary.
SUS-03	Achei o produto fácil de utilizar.	I thought the product was easy to use.
SUS-04	Acho que necessitaria de ajuda de um técnico para conseguir utilizar este produto.	I think that I would need the support of a technical person to use this product.
SUS-05	Considerei que as várias funcionalidades deste produto estavam bem integradas.	I found that the various functions of this product were well integrated.
SUS-06	Achei que este produto tinha muitas inconsistências.	I thought there was too much inconsistency in this product.
SUS-07	Suponho que a maioria das pessoas aprenderia a utilizar rapidamente este produto.	I imagine that most people would learn to use this product quickly.
SUS-08	Considerei o produto muito complicado de utilizar.	I found the product very cumbersome to use.
SUS-09	Senti-me muito confiante a utilizar este produto.	I felt very confident using this product.
SUS-10	Tive que aprender muito antes de conseguir lidar com este produto.	I needed to learn a lot before I could use this product.

C.2.3 Study-Specific Statements

The five study-specific statements used the same response format as the SUS items: 1 represented *Discordo totalmente* and 5 represented *Concordo totalmente*. These statements were analysed separately and were not included in the SUS score.

Table C.4: Study-specific statements used in the final questionnaire.

Portuguese wording used	English translation
As perguntas do wizard foram claras.	The wizard's questions were clear.
Os ecrãs de revisão permitiram verificar se o resultado correspondia ao que pretendia.	The review screens allowed me to verify whether the result corresponded to what I intended.
Senti-me capaz de identificar e corrigir informação incorreta.	I felt able to identify and correct incorrect information.
Compreendi o que seria importado para o DISME.	I understood what would be imported into DISME.
Consideraria útil usar esta ferramenta para produzir o primeiro rascunho de um sistema.	I would consider this tool useful for producing the first draft of a system.

C.2.4 Open Questions

Table C.5: Open questions used in the final questionnaire.

Portuguese wording used	English translation	Required
Qual foi a parte mais difícil e porquê?	What was the most difficult part, and why?	Yes
Em que momento, incluindo durante a exploração livre, teve mais dúvidas ou menos confiança no resultado? Justifique.	At what point, including during free exploration, did you have the most doubts or the least confidence in the result? Explain your answer.	No
O que alteraria no wizard para tornar o processo mais claro ou útil?	What would you change in the wizard to make the process clearer or more useful?	Yes

C.2.5 Remote-Access Condition

The questionnaire asked: *Em que medida a ligação remota através do TeamViewer interferiu com a sua utilização?* (To what extent did the remote connection through TeamViewer interfere with your use?). The available responses were *Não interferiu*, *Interferiu ligeiramente*, *Interferiu moderadamente*, *Interferiu muito*, and *Não consigo avaliar*. Participants could optionally explain how the connection had interfered. This response was used only to describe the remote-access condition and was not included in the SUS score or the study-specific statements.

SYSTEM WALKTHROUGH

D.1 Start with the business description

This appendix follows the full CONFERENCE example introduced in Chapter 5. It shows the normal-mode workflow from an initial description to the preparation and execution of *Submit Review*, using *Review* and *Assignment* to illustrate the detailed checks.

The earlier figures combine preserved interface captures with screens reopened from documented states of the example. The final Form and result captures come from a separate execution of the reviewed *Submit Review* rule in an isolated demonstration database. Together, they document the procedure and its stored result. Model names are in English; the wizard controls retain their Portuguese labels.

1. Open the wizard in normal mode. For the later import stages, have the DISME API address and an account authorised to import models and save Action Rules. Execution will also require a reviewer account and application configuration.
2. Describe the organisation, the intended users and the problem. The first answer in the original conversation was, translated into English:

We organise scientific conferences. The system will be used by authors, reviewers, and the conference chair to manage submissions, reviews, and decisions in one place.

Continue answering the questions in the response box. Explain submission, reviewer assignment, review and decision handling, including deadlines and visibility. Distinguish an article, its assignment to a reviewer and the review produced for that assignment; these records serve different purposes.

Check before continuing. The description identifies the users, the main operations and the information they need. The next stage is a proposal to review, so corrections can still be made before the detailed models are built.

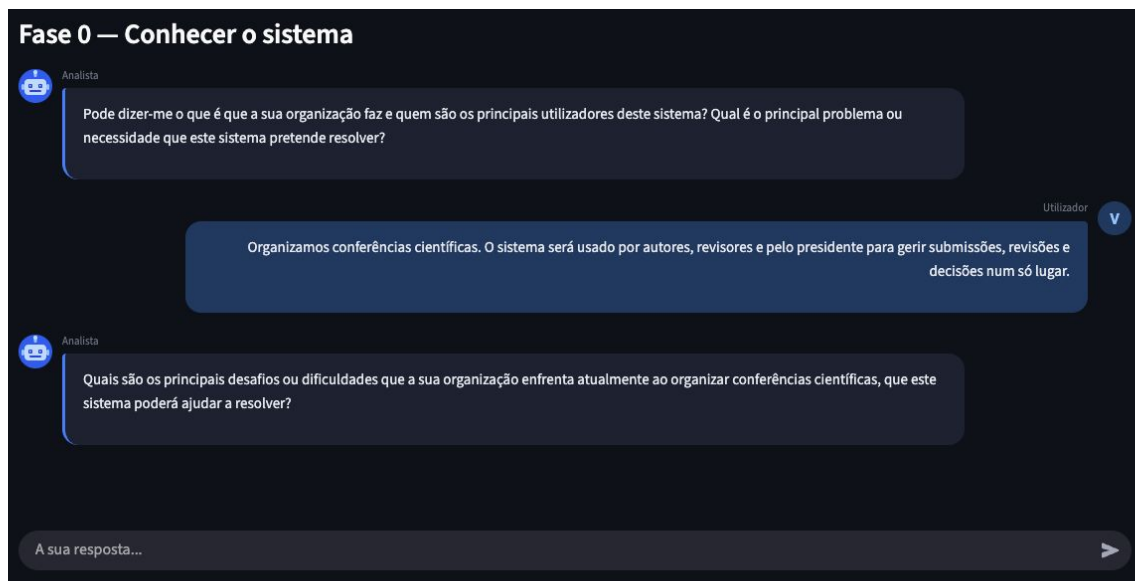


Figure D.1: Opening exchange from the preserved conversation: the business description and the next question.

D.2 Inspect the proposal and request a correction

The overview is the first opportunity to compare the extracted scope with the intended system. The initial six-entity proposal is illustrated below with English labels.



Figure D.2: Initial entity proposal, including Decision as a separate entity.

1. Read each entity description and the proposed tasks. In this example, *Decision* is removed as a separate entity; the article retains the decision information. The scope also needs conference configuration, authorship links and conflict declarations.
2. In **O que pretende corrigir?**, state the changes together. Select **Preparar correção** to obtain a proposal for review. The continuation button is unavailable while a correction request is pending.

O que pretende corrigir?

Remove Decision as a separate entity. Add Configuration for deadlines and the current phase, Authorship to link articles to their authors, and Conflict Declaration to record conflicts between reviewers and authors.

Preparar correção

Está correto — continuar

Prepare e reveja a correção, ou apague o texto para confirmar a proposta atual.

Figure D.3: Correction request specifying the removal and the three additions, before preparing the revised proposal.

Check before continuing. Describe what each added entity represents. Merely asking for three extra names would leave their meaning and relationships unclear.

D.3 Accept the correction and confirm the scope

Preparing a correction does not immediately replace the current proposal. Read the revised entity list and check that the requested changes preserve the rest of the system.

1. Check that *Configuration* stores conference settings, *Authorship* links an article to an author, and *Conflict Declaration* represents a conflict involving a reviewer and an author. Confirm that *Article*, *Author*, *Reviewer*, *Review* and *Assignment* remain.
2. Select **Aceitar correção** if the revision is correct. Select **Manter versão anterior** if it is not, then reformulate the request. Once the accepted overview is correct, select **Está correto - continuar** to begin Phase 1.

Check before continuing. The accepted scope contains eight entities. This is the reviewed reference example, not a guarantee that a fresh generation will return the same proposal.

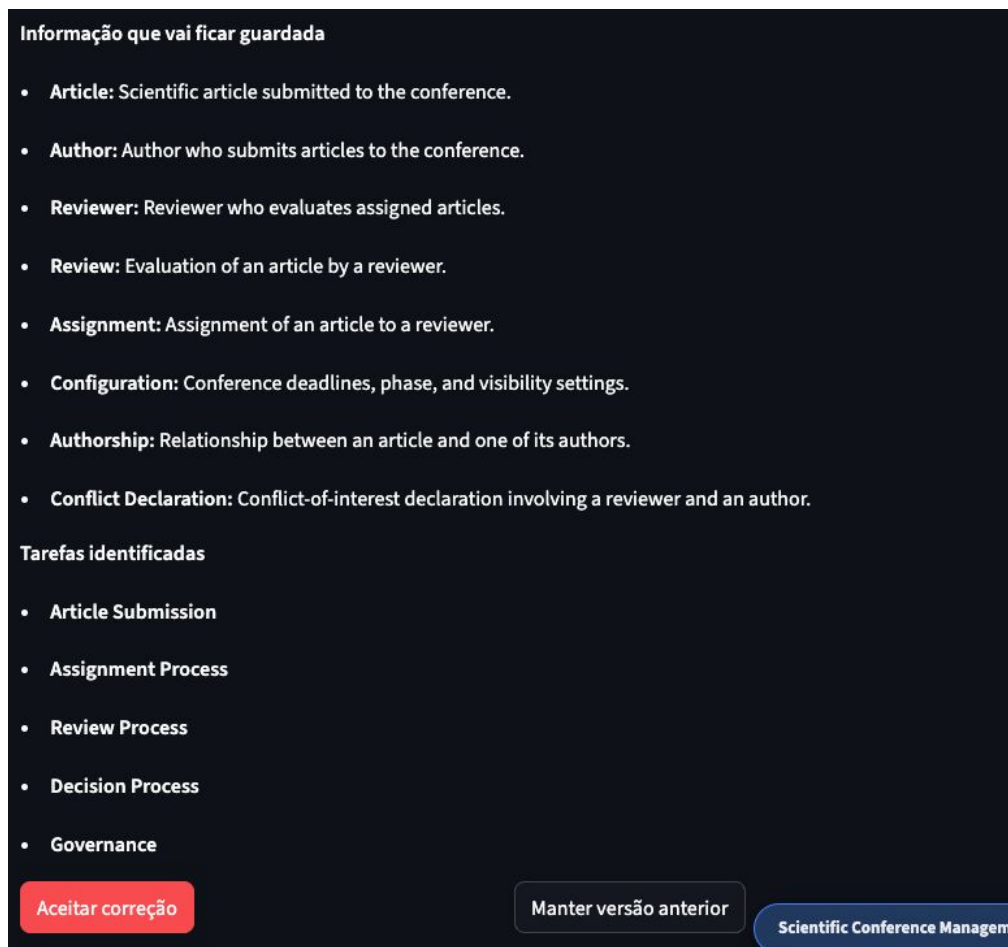


Figure D.4: Revised eight-entity proposal with the controls to accept it or retain the previous version.

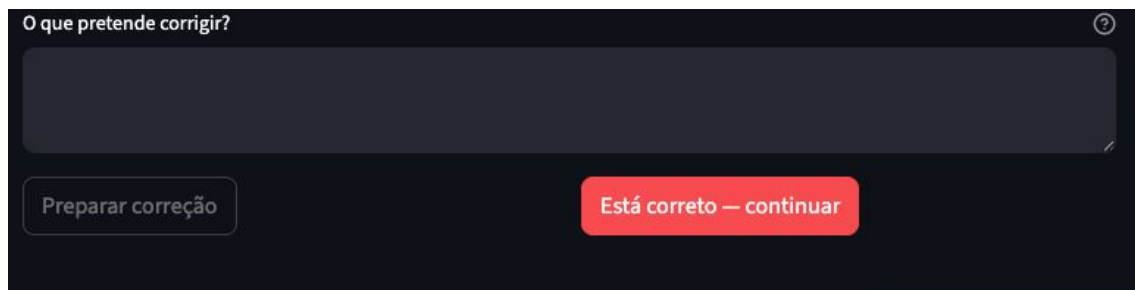


Figure D.5: After accepting the correction, the overview can be confirmed and the wizard can continue.

D.4 Define and review the State Model

1. In Phase 1, answer the questions for each entity about properties, value types, permitted values and relationships. For *Review*, specify Score, Status, Recommendation, Comments, Reviewer and Article.
2. Use an integer for Score. Set Recommendation to Accept, Reject, Minor revision or Major revision, and Status to Pending or Submitted. Comments is optional. Article and Reviewer refer to existing records, rather than storing duplicate names as free text.
3. Review each entity before confirming it. At the final review, expand *Review* and inspect the properties together. A reference must target the intended entity and display a suitable property, such as Article.Title or Reviewer.Name.

Fase 1 concluída — Revisão Final

8 entidades, 33 campos extraídos.

- > Article (6 campos)
- > Author (3 campos)
- > Reviewer (3 campos)
- > Review (6 campos)

Evaluation of an article by a reviewer.

Criada por: Submit Review

Campo	Tipo	Obrig.	Editável	Detalhe
Score	Número inteiro	sim	sim	
Status	Escolha de uma opção	sim	sim	Pending, Submitted
Recommendation	Escolha de uma opção	sim	sim	Accept, Reject, Minor revision, Major revis
Comments	Texto	não	sim	
Reviewer	Escolha de informação guardada	sim	não	-> Reviewer (Name)
Article	Escolha de informação guardada	sim	não	-> Article (Title)
- > Assignment (5 campos)
- > Configuration (5 campos)
- > Authorship (2 campos)
- > Conflict Declaration (3 campos)

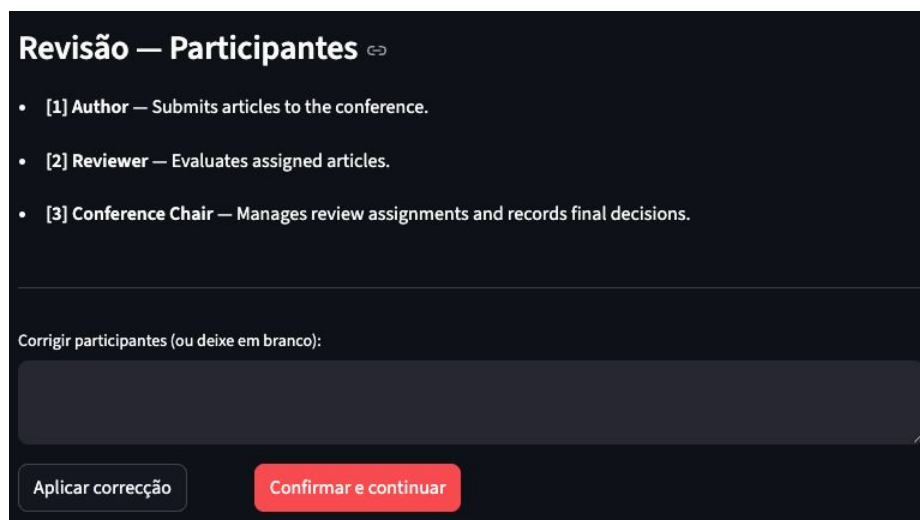
Figure D.6: Final State Model review. Review contains four ordinary properties and two references to existing entities.

Also expand *Assignment* and check its Article and Reviewer references: the execution later obtains these values from the selected Assignment. Use the global correction box if the complete model still needs a change, then select **Continuar para Fase 2**.

Check before continuing. The English reference model contains eight entity types and thirty-three properties. Counts are a useful consistency check; types, allowed values and reference targets still require inspection.

D.5 Confirm roles and the Process Model

1. In Phase 2, identify who performs the operations. Review the roles *Author*, *Reviewer* and *Conference Chair*. If a responsibility is incorrect, describe the correction and apply it; otherwise select **Confirmar e continuar**.



Revisão — Participantes ↻

- [1] **Author** — Submits articles to the conference.
- [2] **Reviewer** — Evaluates assigned articles.
- [3] **Conference Chair** — Manages review assignments and records final decisions.

Corrigir participantes (ou deixe em branco):

Aplicar correção Confirmar e continuar

Figure D.7: Role review with the three responsibilities and the correction and confirmation controls.

2. For each process, explain its transactions, who initiates and executes them, their results, and which transactions start or end the process. For *Submit Review*, the reviewed model uses *Reviewer* for both roles and *Review* as the result.
3. At the final review, expand the processes and check these associations. The complete reference has five processes and thirteen transactions, including the auxiliary registration transactions. When the structure is correct, select **Importar para o DISME**.

Check before continuing. Each transaction is attached to the intended process and result entity. A correct process name alone does not establish that its responsibilities and result are correct.



Figure D.8: Final Process Model review with Review Process expanded: Submit Review, its roles, result and start/end flags.

D.6 Import the structural models

The structural import makes the reviewed entity, property, role, process and transaction records available in DISME. It precedes the query and rule stages because those stages must refer to the imported records.

1. On the import screen, resolve any validation errors before continuing. Confirm the API address and use credentials for the intended DISME instance. The local address in the figure belongs to the documentation environment.
2. Select **Iniciar importacao** once and wait for the result. If the wizard reports a failure, inspect the reported cause before retrying. A generated model is not yet an imported model.

The completion screen below is restored from the saved successful import of the English reference model. It provides the transition to the query stage.

Imported structure	Reference totals
Entity types and properties	8 entity types; 33 properties.
Roles and processes	3 roles; 5 processes.
Transactions	13 transaction types.



Figure D.9: Structural import screen, with the API connection and import control.



Figure D.10: Saved import completion state and the control to continue to Phase 3.

Check before continuing. Confirm the import result, then select **Continuar para Fase 3**. These totals describe the reference import; they do not imply that all behavioural rules have already been created or activated.

D.7 Review form-choice and logic queries

The two query stages serve different needs. Phase 3 restricts the records offered by reference fields. Phase 3b supplies searches required by transaction behaviour.

1. In Phase 3, specify that the Article field of *Assignment* should offer submitted articles. Check the target entity Article, the Status property, the equality operator and the fixed value Submitted. Import the reviewed definitions with **Importar queries para o DISME**.



Figure D.11: SubmittedArticles restricts the Article choices available when creating an Assignment.

2. In Phase 3b, explain which existing records a transaction needs to find. The illustrated query searches for pending Assignments for a particular Article and Reviewer. Article and Reviewer are supplied from the execution context; Pending is a fixed Status value.
3. Check the filters and parameter bindings before selecting **Importar queries de lógica para o DISME**. Continue after the import succeeds. A transaction that only reads its existing context need not acquire an unnecessary additional query.

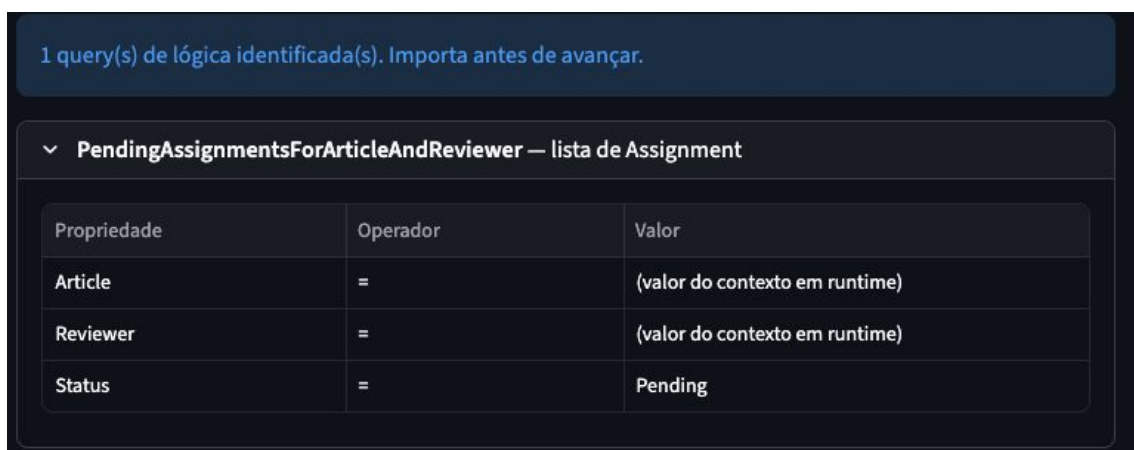


Figure D.12: Logic query with Article and Reviewer as runtime parameters and Status fixed to Pending.

Check before continuing. These are reviewed query definitions, not execution results.

The reviewer application list is configured separately: its filter must use the Reviewer associated with the signed-in account.

D.8 Specify and inspect the Submit Review rule

Phase 4 works through the transaction types. The detailed example below is *Submit Review*, shown as transaction 6 of 13. Explain the form fields, validations, automatic values, changes to existing records and completion message.

1. Request required Score and Recommendation fields and optional Comments. Require an integer Score between 1 and 5, inclusive. Explain that Article and Reviewer come from the selected Assignment.
2. Specify Review.Status = Submitted and Assignment.Status = Completed. Inspect the explanation and individual actions, paying particular attention to which entity each assignment modifies.

Fase 4 — Submit Review (6/13)

Explicação da regra gerada:

When Submit Review reaches the Executed state, the wizard asks for the score, recommendation, and optional comments. It links the new Review to the Reviewer and Article from the Assignment, marks the Review as Submitted and the Assignment as Completed, and displays a confirmation.

Quando 'Submit Review' chega a Executada:

- Formulário (criar): **Submit Review [3] — Review Form**
- Score *
- Recommendation *
- Comments
- Atribuição automática: **Reviewer** = Assignment.Reviewer
- Atribuição automática: **Article** = Assignment.Article
- Atribuição automática: **Status** = "Submitted"
- Atribuição automática: **Status** = "Completed"
- Mostra uma notificação: "Review submitted successfully."

Figure D.13: Submit Review review screen: form fields, references from Assignment, status changes and notification.

Check before continuing. Verify the score limits and both reference bindings, then select Confirmar e continuar when the rule is correct. The next section explains the available correction controls.

D.9 Correct form fields or request a behaviour change

The per-transaction review offers two ways to correct a rule. The manual editor changes the Form fields and required flags. The change-request box expresses changes to the transaction behaviour.

1. Expand **Editar campos manualmente**. Check the Form name and the included fields. Leave Score and Recommendation marked as required and Comments unmarked. Use the removal and addition controls only if a field is missing or unnecessary.
2. Select **Aplicar edição** after changing the fields. Reopen the resulting preview and check that the form still matches the intended input.

▼ Editar campos manualmente

Nome do formulário

Submit Review [3] — Review Form

Campos incluídos:

Score	☒ Obrigatório	☐ Remover
Recommendation	☒ Obrigatório	☐ Remover
Comments	☐ Obrigatório	☐ Remover

Adicionar campo

— ▼

Aplicar edição

Figure D.14: Manual field editor: Score and Recommendation are required; Comments is optional.

Intended change	What to use and verify
Field inclusion or requirement	Use the manual field editor; verify the resulting field list and required flags.
Validation or automatic assignment	Describe the change in the request box, use Aplicar alteração, and inspect the resulting behaviour.
No further change	Use Confirmar e continuar after checking the complete rule.

Check before continuing. A required flag does not define the valid score range. Check the integer type and the inclusive limits separately, including in the imported rule blocks.

D.10 Approve the reviewed rules

After the transaction interviews, the final review separates rules that are pending, approved or flagged. The screenshots focus on the single Submit Review rule used in this illustration; the same procedure applies to the remaining generated rules.

1. Open each pending rule and compare its actions with the intended behaviour. Select **Aprovar** only after checking the form and assignments. Use **Sinalizar problema** for a rule that still needs correction.
2. Resolve pending and flagged rules before import. The disabled import control and the message below explain the approval requirement.

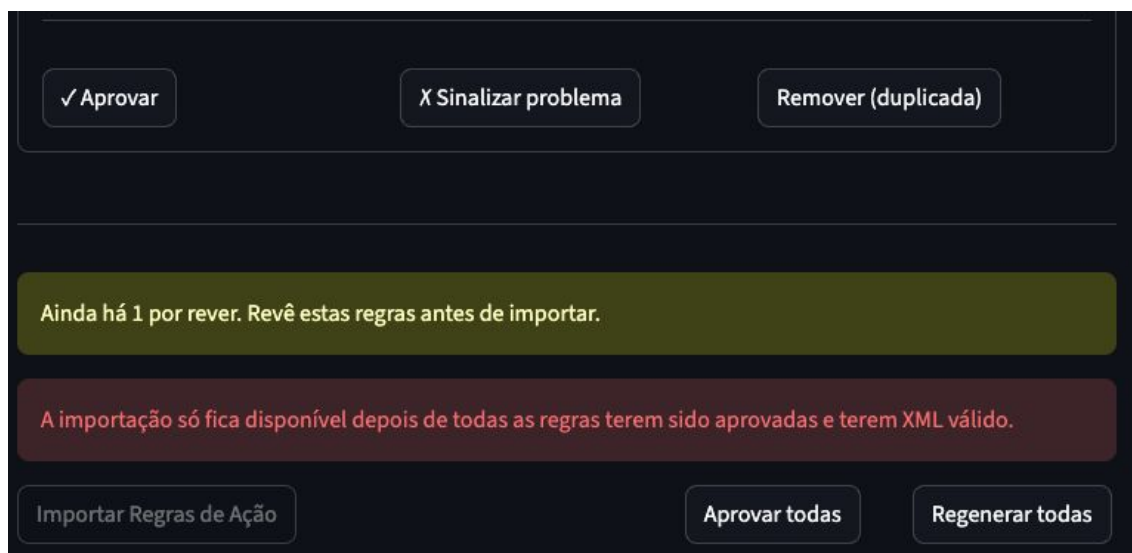


Figure D.15: Pending review: approval and problem-reporting controls, with import unavailable.

After approval, verify that there are no remaining rules to review. The final summary should agree with the rule set being imported. The one-rule summary below belongs to the focused illustration, not to all thirteen transactions.



Figure D.16: Approved Submit Review rule: the import action is now available.

Check before continuing. Approval in the wizard permits draft import. The rule still needs to be loaded and saved in the DISME Action Rules Manager before it becomes available for execution.

D.11 Import the draft and activate it in DISME

1. Select **Importar Regras de Ação** to open the draft import screen. Check the target API and account again, then select **Importar rascunhos**. Confirm that the required drafts were imported.
2. Open the **Action Rules Manager** in DISME. Choose **Load Draft** and select the draft associated with *Submit Review*. Loading the draft should display its blocks in the editor.

Use the editor controls shown below to load the draft and subsequently save it as an active rule. Review the blocks first, following the checks in the next section.

Step	What it makes available
Import rascunhos	A draft that can be opened in the rule editor.
Load Draft	The draft blocks for inspection and editing.
Save Action Rule	The active Action Rule used during execution.

Check before continuing. Do not treat the draft-import confirmation as proof that the rule is active. Complete the editor review and wait for the Save Action Rule confirmation.



Importar Rascunhos de Regras para o DISME

1 regra(s) prontas para importar.

URL da API DISME

http://localhost:8001/api

Email

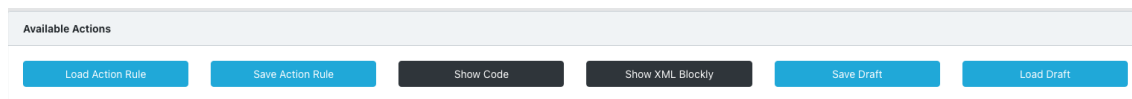
admin@admin.com

Palavra-passe

.....

Importar rascunhos

Figure D.17: Wizard screen for importing the reviewed rule as a draft.



Available Actions

Load Action Rule Save Action Rule Show Code Show XML Blockly Save Draft Load Draft

Figure D.18: Action Rules Manager controls. Load Draft opens the imported draft; Save Action Rule saves the active rule.

D.12 Check the rule blocks before saving

In the imported blocks, verify both the input Form and the effects on records. The two close views below are taken from the same Submit Review rule in the DISME editor.

1. Check that the Form creates a Review, requires Score and Recommendation, and leaves Comments optional. Confirm the integer type and the two inclusive score conditions: greater than or equal to 1, and less than or equal to 5.
2. Check the automatic assignments. Review.Reviewer must come from Assignment.Reviewer, and Review.Article from Assignment.Article. The new Review receives Submitted; the selected Assignment receives Completed.
3. After checking the complete rule, select **Save Action Rule** and wait for confirmation. Repeat this activation step for the other rules required by the application.

The figure shows a configuration for a 'Submit Review' form block. It is set to be executed when the 'Submit Review' action is performed. The form contains several fields and validation rules:

- Action:** 'create instance of a single entity type' with name 'Submit Review [3] - Review Form' and entity type 'Review'.
- Form Fact:** 'Score' (int, mandatory). It has two validation conditions: 'Higher Equal' with value 1 and 'Less Equal' with value 5.
- Form Fact:** 'Recommendation' (enum, mandatory).
- Form Fact:** 'Comments' (text, not mandatory).

Figure D.19: Form block with the required fields and the two score limits.

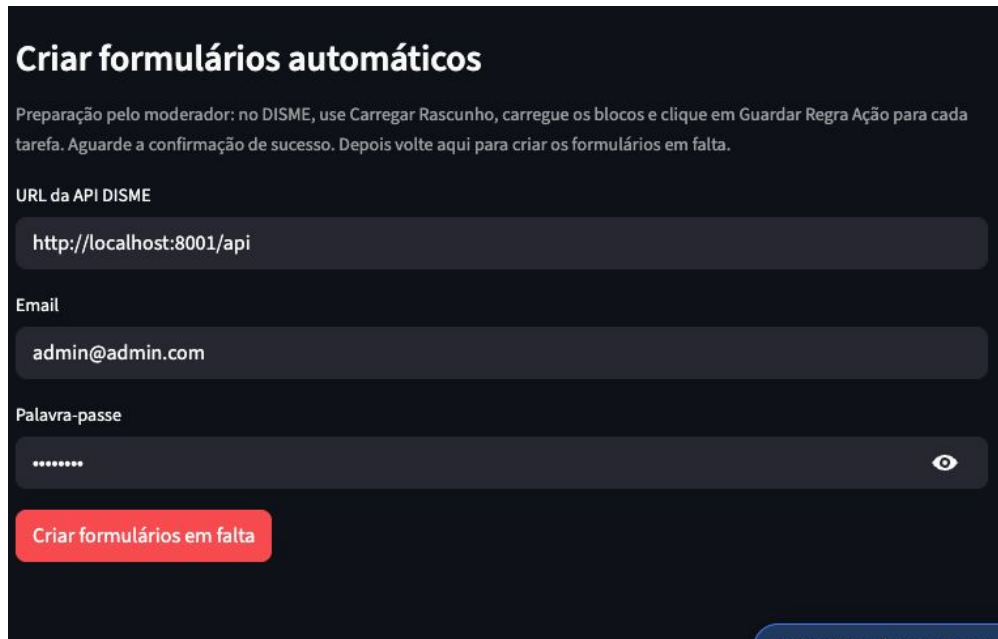
The figure shows four 'assign expression' actions that automatically bind the 'Review' entity to the 'Assignment' entity and set their statuses:

- Action 1:** Assigns the 'Reviewer' property of the 'Review' entity to the 'Reviewer' property of the 'Assignment' entity.
- Action 2:** Assigns the 'Article' property of the 'Review' entity to the 'Article' property of the 'Assignment' entity.
- Action 3:** Assigns the 'Status' property of the 'Review' entity to the 'Submitted' property value of the 'Assignment' entity.
- Action 4:** Assigns the 'Status' property of the 'Assignment' entity to the 'Completed' property value of the 'Assignment' entity.

Figure D.20: Automatic assignments bind the Review to the selected Assignment and set the two records to different statuses.

D.13 Prepare the Form and the reviewer application

1. Return to the wizard after saving the active rule. Under **Criar formulários automáticos**, select **Criar formulários em falta**. This step checks the eligible active rules and prepares any missing Forms.



Criar formulários automáticos

Preparação pelo moderador: no DISME, use Carregar Rascunho, carregue os blocos e clique em Guardar Regra Ação para cada tarefa. Aguarde a confirmação de sucesso. Depois volte aqui para criar os formulários em falta.

URL da API DISME

Email

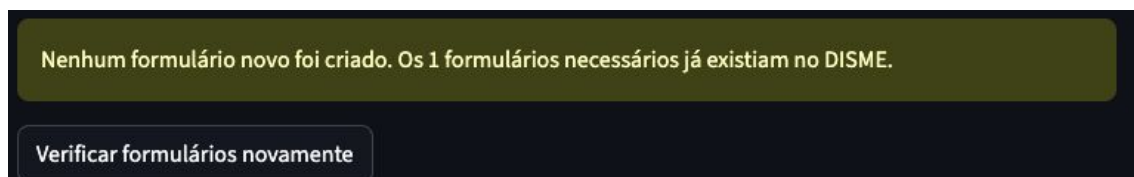
Palavra-passe



Criar formulários em falta

Figure D.21: Form preparation after the drafts have been imported and saved as active rules in DISME.

Read the result before continuing. This earlier saved example reports one eligible Form, no new Form and one existing Form retained. The separate final execution used a fresh instance, where the required Form was created.



Nenhum formulário novo foi criado. Os 1 formulários necessários já existiam no DISME.

Verificar formulários novamente

Figure D.22: Saved result: the required Form already existed, so no new Form was created.

2. In DISME, an administrator prepares the reviewer UserApp, permissions and domain records. Create an Article, a Reviewer and a pending Assignment linking them. Give the execution account the Reviewer role and associate it with the Reviewer record.
3. In **Apps Management**, prepare the application screen and list component. Configure `PendingAssignmentsForCurrentReviewer` to return pending Assignments for the

signed-in account. For result inspection, also provide lists of submitted Reviews and completed Assignments.

4. Configure the list-row action to start *Submit Review* for the selected Assignment. The selected record must be carried into the execution context; otherwise the rule cannot reliably obtain the intended Article and Reviewer.

Check before continuing. The rule, Form, account association and contextual list action are separate prerequisites. The wizard import alone does not establish that the reviewer application is ready.

D.14 Submit a review and verify the stored result

1. Sign in with an account assigned to the Reviewer role and open the prepared application. Locate the intended pending Assignment and start *Submit Review* from that row. Confirm that the selected Assignment refers to the intended article.
2. Complete the Form. In this execution, the values are Score = 4, Recommendation = Accept and Comments = Clear methodology. Check the values and select **Submit**.

The screenshot shows a web application interface with a modal form titled "Fill the following form" overlaid on a task list. The form contains three input fields: "Score" with the value 4, "Recommendation" with the value Accept, and "Comments" with the value Clear methodology. A blue "Submit" button is at the bottom of the form. The background shows a task list with a "Pending tasks 1" header and a "Start Task for a" button. There are also "Delegations 0" and "Execute Pending Task of an Existing Process" buttons visible.

Figure D.23: The actual Submit Review Form immediately before submission, with Score 4, Recommendation Accept and the comment entered.

After the operation completes, inspect the saved records and refresh the application list. Apply the checks below. The next section shows the records obtained from this submission.

Where to check	Expected result after successful submission
Review	One new Review stores the entered Score, Recommendation and Comments. Its Article and Reviewer are those of the selected Assignment, and its Status is Submitted.
Assignment	The same selected Assignment has Status = Completed.
Reviewer list	The completed Assignment disappears from the pending list after refresh.
Task or process	The transaction has completed without an unresolved execution error.

If the Form opens but the references or status changes are incorrect, first inspect the selected Assignment context and the saved rule. If the operation appears successful but the item remains listed, check the stored Assignment status and the list filter.

Check before continuing. The submitted values, both references, the Assignment update and the refreshed list must agree. Record identifiers allow the before and after states to be traced.

D.15 Inspect the records created and updated

The final execution started from one pending Assignment (entity 3), linking Reviewer 2 to Article 1, with no Review yet stored. Submitting the Form created one Review (entity 4) and updated that same Assignment. The record identifiers were checked in the API response and database.

Review created



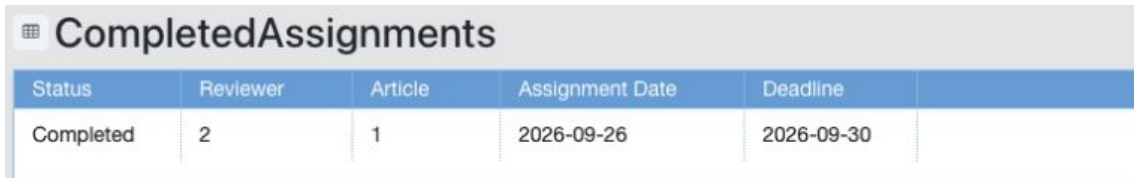
SubmittedReviews					
Status	Score	Recommendation	Comments	Reviewer	Article
Submitted	4	Accept	Clear methodology.	2	1

Figure D.24: The submitted Review stores Score 4, Recommendation Accept, the entered comment and Status Submitted. Reviewer 2 and Article 1 match the selected Assignment.

Open **Submitted reviews** and compare the stored values with those entered in the Form. The displayed references are entity identifiers: **2** is the Reviewer and **1** is the Article. Both were obtained from the selected Assignment during execution.

Assignment completed

Open **Completed assignments** to check the status update. Then return to **Pending assignments** and refresh: the completed item is no longer returned. In this execution, the pending list changed from one row to zero rows, and Review Process 4 reached the finished state.



Status	Reviewer	Article	Assignment Date	Deadline
Completed	2	1	2026-09-26	2026-09-30

Figure D.25: The selected Assignment now has Status Completed and retains the same Reviewer and Article references. Its assignment date and deadline are also shown.

These results confirm the stored effects of this Submit Review execution: a new Review with the submitted values, matching references and the completed Assignment. This demonstration uses the prepared example and application configuration described above.

DISME'S EBNF GRAMMAR

Table I.1: EBNF Grammar Definition from [8]

Element	Definition
when	WHEN transaction_type IS HAS_BEEN transaction_state { action }
transaction_type	STRING NOTE: has to be a transaction_type specified in the system.
transaction_state	INITIAL REQUESTED PROMISED EXECUTED DECLARED ACCEPTED DECLINED REJECTED STOP QUIT REVOKE_REQUEST_REQUESTED REVOKE_REQUEST_ALLOWED REVOKE_REQUEST_REFUSED REVOKE_PROMISE_REQUESTED REVOKE_PROMISE_ALLOWED REVOKE_PROMISE_REFUSED REVOKE_DECLARE_REQUESTED REVOKE_DECLARE_ALLOWED REVOKE_DECLARE_REFUSED REVOKE_ACCEPTANCE_REQUESTED REVOKE_ACCEPTANCE_ALLOWED
action	causal_link assign_expression user_input edit_entity_instance user_output produce_doc if while foreach API_CALL
user_output	STRING NOTE: special HTML code defined in the template manager component, dashboard will output this to the user interface.
produce_doc	static_template form_template
static_template	STRING NOTE: special rendered HTML code annotated with custom directives from which a PDF is generated.

form_template	STRING NOTE: special rendered HTML code annotated with custom directives from which a PDF is generated. The interpretation of some directives prompts the end-user at runtime for additional input.
assign_expression	property "=" term property_value
property	STRING NOTE: has to be an existent property specified in the system.
causal_link	transaction_type MUST BE transaction_state [min max] [CANCEL_PROC] [CONTINUE_IF_SAME_USER] NOTE: min max are optional and by default come with 1 as pre-filled; if min doesn't exist, by default = 1. If max doesn't exist, by default = min. Cancel_proc refers to whether the causal_link cancels the current process. Continue_if_same_user marks whether the execution engine should take the user directly to the execution of the causal_link task.
min	Integer
max	Integer *
user_input	{ form_property }-
edit_entity_instance	{ entity_detail } { form_property form_ent_type }- NOTE: properties inserted here must have the flag 'editable' as true.
form_property	property [form_calculation] [enable_condition] { validation_condition }- [MANDATORY] NOTE: mandatory is a checkbox in the block that specifies if the filling of the property should be mandatory in its respective form.
form_ent_type	LINKING_PROPERTY { form_property }- NOTE: one_to_many or many_to_many ent_types only. Linking_property refers to which property is assigned automatically by the engine.
entity_detail	property NOTE: properties specified here are to be shown in the entity selection modal select box in 'edit entity instance' actions.
form_calculation	compute_expression NOTE: used when we want that the value assigned to a property is automatically computed based on some expression.

<i>enable_condition</i>	ENABLE condition NOTE: this is used when we want that a property is “hidden/disabled” from the form unless the specified condition is true.
<i>validation_condition</i>	[NOT] validation_condition_type [EXTRA_FIELD_1] [EXTRA_FIELD_2] [user_output] NOTE: The not, extrafield1 and extrafield2 fields will appear depending on the validation_condition_type chosen.
<i>validation_condition_type</i>	REQUIRED IS_NUMBER IS_INTEGER EQUAL_TO MAX_WORD_LENGTH LESS_EQUAL HIGHER_EQUAL HIGHER_THAN LESS_THAN MIN_LENGTH BELONGS_RANGE MAX_LENGTH MIN_WORD_LENGTH MAX_WORD_LENGTH HAS_CHARACTER HAS_WORD IS_EMAIL IS_URL CUSTOM_VALIDATION REG_EXPRESSION
<i>if</i>	IF condition THEN { action }- [ELSE { action }-]
<i>condition</i>	(ISTRUE NOT evaluated_expression condition) (AND OR { evaluated_expression condition }-) NOTE: if condition is of type ISTRUE engine will evaluate the expression; if condition is of type NOT, engine will evaluate either the expression or condition; if it is of type AND or OR it will accordingly evaluate the specified expressions/conditions.
<i>evaluated_expression</i>	comp_evaluated_expression user_evaluated_expression
<i>comp_evaluated_expression</i>	term logical_operator term property_value
<i>user_evaluated_expression</i>	STRING NOTE: this is a simple text input and dashboard shows this “textual informal expression” that has to be evaluated by the user who will decide on a result of true or false.
<i>logical_operator</i>	“<” “>” “==” “!=”
<i>property_value</i>	STRING NOTE: must be a possible value of a property with 2 cases: 1) value_type is enum and one can select all allowed values for the selected property 2) value_type is prop_ref where possible values to select will be the values associated with the property fk_property_id.
<i>term</i>	constant value property query compute_expression produce_doc
<i>constant</i>	value_type STRING NOTE: Can be a constant defined on the system, or can be a new specification of a constant.

<i>value</i>	value_type STRING NOTE: free value inserted when editing the Action Rule. Must also specify the value type when specifying the new value.
<i>value_type</i>	TEXT INTEGER_NUMBER REAL_NUMBER BOOLEAN ENUM DATE TIME
<i>query</i>	STRING { term } NOTE: has to be an existing query specified in the query table. It will have terms if it's a dynamic query where the user can specify query parameters.
<i>compute_expression</i>	term { compute_operator term }-
<i>compute_operator</i>	"+" "-" "x" "/" "^"
<i>while</i>	WHILE condition { action }-
<i>foreach</i>	FOREACH set { action }-
<i>set</i>	SET_OF_ELEMENTS
<i>constant</i>	value_type STRING NOTE: Can be a constant defined on the system, or can be a new specification of a constant. In the latter case, the name, value and value type of the constant to be created must also be specified.
<i>value</i>	value_type STRING NOTE: free value inserted when editing the Action Rule. Must also specify the value type when specifying the new value.
<i>value_type</i>	TEXT INTEGER_NUMBER REAL_NUMBER BOOLEAN ENUM DATE TIME



2026

Generation of Business Models in a Low-Code Platform with Large Language Models

Vasco Nunes

