MySirt Execution Engine

Architecture Overview of the

MySirt Model Execution Engine

Technical Documentation
Version 2.0 – August 2026

Prepared by
MySirt Team

Primary Author
Leda Santosuosso

Document Information

Document	MySirt Model Execution Engine Architecture Overview
Product	MySirt
Version	2.0
Date	August 2026
Document Maintainer	Leda Santosuosso
Status	Technical Reference

Prerequisites

This document assumes familiarity with the overall MySirt architecture and with the fundamental principles of the MySirt Domain-Specific Language (DSL).

Before reading this document, it is recommended to read:

- **MySirt Ecosystem**
- **MySirt DSL Design Principles**

These documents introduce the MySirt ecosystem, the role of the DSL as the single source of truth, and the architectural principles on which the Model Execution Engine operates.

1. Introduction

Relationship with MySirtApp2

Within the MySirt ecosystem, the Model Execution Engine is the component responsible for transforming declarative application models into Executable Information Systems. Its architectural foundation is the MySirt DSL and its execution pipeline.

The current product includes MySirtApp2, a modeling and management environment designed to simplify the creation, maintenance and deployment of MySirt systems.

MySirtApp2 provides visual modeling, workspace management, versioning and deployment facilities. These capabilities improve the authoring experience but do not alter the underlying architectural principles of the MySirt Execution Engine.

Internally, MySirtApp2 maintains a workspace representation that is automatically transformed into the MySirt DSL before generation. The DSL therefore remains the canonical input processed by the Model Execution Engine, while the workspace representation belongs exclusively to the authoring environment.

1.1 Purpose and Scope

This document provides an architectural overview of the **MySirt Model Execution Engine**, the component responsible for transforming declarative application models into Executable Information Systems.

Building on the concepts introduced in the *MySirt Ecosystem* and *MySirt DSL Design Principles* documents, this overview focuses on the responsibilities of the Execution Engine and on the architectural process through which a DEF model is transformed into an executable application.

In particular, this document describes:

- the role of the Model Execution Engine within the MySirt architecture
- the processing pipeline from DEF model to Executable Information System
- model parsing, validation and transformation
- the relationship between generated application artifacts and the shared runtime infrastructure
- the architectural principles that support generation, regeneration and execution

The document focuses exclusively on the **MySirt Model Execution Engine**.

It does not describe in detail:

- modeling tools used to create application models
- the design principles of the MySirt DSL (covered in *MySirt DSL Design Principles*)
- the technical architecture of the MySirt DSL (covered in *Technical Architecture of the MySirt DSL*)
- the architecture and runtime behavior of generated applications (covered in *Architecture of a MySirt Generated System*)

2. System Generation Pipeline

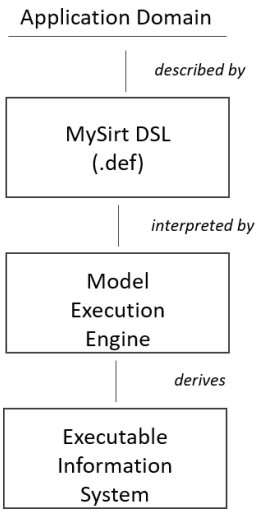

Figure 1 – From DEF model to Executable Information System

The MySirt Model Execution Engine operates on **DEF models** expressed in the MySirt Domain-Specific Language (DSL).

The DEF model constitutes the canonical input of the generation pipeline described in this chapter. It provides the complete declarative specification of the information system to be derived by the Model Execution Engine, while remaining independent of its technical implementation.

The following sections focus on the responsibilities of the Model Execution Engine and describe how it transforms a DEF model into an Executable Information System through parsing, validation, model construction and transformation.

2.1 Definition Parsing

The first stage of the generation pipeline is the analysis of the DEF model.

During this phase, the Model Execution Engine reads the definition expressed in the MySirt DSL, identifies the language constructs declared in the model—such as entities, fields, relationships, forms, triggers and other declarative elements—and performs syntactic validation to verify that the model conforms to the MySirt DSL grammar.

Only syntactically valid models can proceed to the following stages of the generation process.

2.2 Internal Model Construction

After successful parsing, the Model Execution Engine builds an internal object model representing every element defined in the DEF model.

This internal representation becomes the authoritative in-memory model of the application and serves as the basis for all subsequent processing performed by the Engine.

Before any transformation takes place, the internal model undergoes semantic validation to verify the consistency of the modeled system. Validation rules include checks on model relationships, attribute compatibility, mandatory references and other architectural constraints. Depending on their severity, validation issues may either prevent system generation or be reported as warnings.

Once validated, the internal model becomes the central representation from which all software artifacts are subsequently derived.

2.3 Model Transformations

Starting from the validated internal model, the Model Execution Engine performs a sequence of model transformations that derive the executable representation of the application.

Rather than processing the textual DEF model directly, all transformation activities operate on the internal object model, ensuring that every generated artifact is derived from a consistent and validated representation of the system.

During this phase, the Engine derives:

- application-specific software artifacts
- the logical database structure
- user interface configurations
- workflow and business logic

- organizational and access-control structures

Each generated artifact preserves the relationships and semantics expressed in the original DEF model, maintaining consistency across all parts of the generated application.

2.4 Executable Information System Derivation

At the end of the transformation process, the Model Execution Engine derives an Executable Information System composed of:

- application-specific software artifacts
- the logical database structure
- interface and process configurations

The resulting Executable Information System operates together with the shared runtime infrastructure provided by MySirt.

Because the DEF model remains the authoritative description of the application, the Executable Information System can be consistently regenerated whenever the model evolves. This regeneration process preserves the alignment between the declarative specification and the derived software artifacts throughout the lifecycle of the application.

The following diagram summarizes the main stages of the MySirt system generation pipeline.

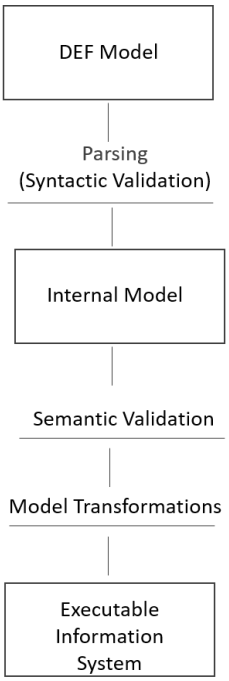

Figure 2 – MySirt System Generation Pipeline

3. Conceptual Dimensions of an Executable Information System

The primary objective of the MySirt Model Execution Engine is to derive an **Executable Information System** from a DEF model.

Although the generated system consists of numerous software artifacts, these artifacts can be conceptually grouped into three complementary dimensions that together describe the operational structure of the application.

These dimensions represent the principal categories of artifacts derived by the Model Execution Engine and correspond to the fundamental aspects required by an information system: data management, application behavior and organizational governance.

The following diagram summarizes these three conceptual dimensions

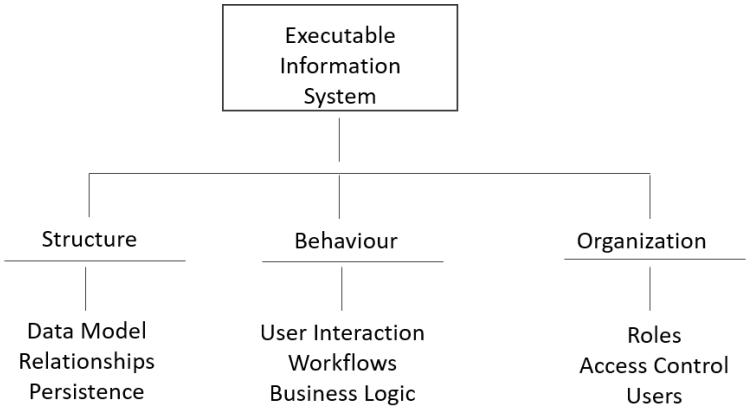

Figure 3 – Conceptual Dimensions of a MySirt Executable Information System

3.1 Structural Dimension

The structural dimension represents the informational foundation of the generated application.

From the entities, attributes and relationships defined in the DEF model, the Model Execution Engine derives the artifacts required to represent and manage the application's information structure, including its logical persistence model.

These artifacts establish the data structures on which all other generated components operate, ensuring that the information model remains consistent with its declarative specification.

3.2 Behavioral Dimension

The behavioral dimension represents the operational logic of the generated application.

From the declarative definitions contained in the DEF model, the Model Execution Engine derives the artifacts that implement user interactions, business processes and application behavior.

This includes forms, triggers, jobs and field-level declarative rules, allowing operational workflows to be expressed as part of the application model rather than being manually implemented throughout the source code.

3.3 Organizational Dimension

The organizational dimension represents the governance structure of the generated application.

From the organizational definitions contained in the DEF model, the Model Execution Engine derives the artifacts responsible for user roles, access rules and authorization policies.

As a result, organizational responsibilities and security constraints become integral parts of the generated information system and remain aligned with the declarative application model.

Runtime Boundary

The MySirt Model Execution Engine derives the application-specific artifacts that constitute an Executable Information System.

However, not every component required for execution is generated. Generic execution capabilities are provided by a shared runtime infrastructure distributed together with every generated application.

This architectural separation allows application-specific behavior to be derived from the DEF model while avoiding the duplication of common execution mechanisms across generated systems.

The following diagram summarizes the conceptual boundary between generated artifacts and the shared runtime infrastructure.

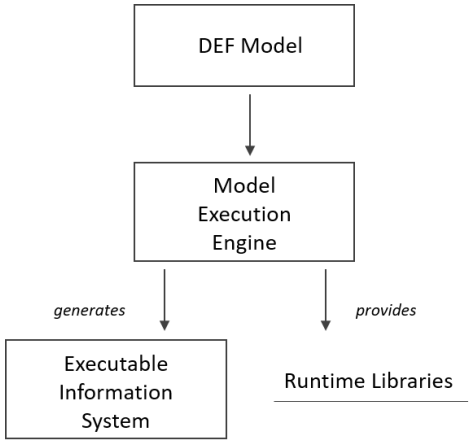

Figure 4 – Conceptual Runtime Architecture of the MySirt Model Execution Engine

4. Regeneration and Extension Boundary

The MySirt Model Execution Engine is designed to preserve regeneration as a fundamental architectural property while allowing controlled application-specific extensions.

To achieve this objective, the generated architecture maintains a clear separation between automatically generated artifacts and manually developed components. This separation allows application-specific behavior to be introduced without modifying the artifacts derived from the DEF model.

For selected application components, the Model Execution Engine derives both a generated implementation and a corresponding extension component intended for manual customization.

During regeneration, only the generated implementation is updated according to the current DEF model, whereas manually developed extensions remain unchanged.

At runtime, the extension component is invoked in preference to the generated implementation whenever customized behavior is available.

This architectural approach preserves the regeneration capability of the system while allowing application-specific extensions to evolve independently of the generated artifacts. As a consequence, the DEF model remains the authoritative specification of the application, while manually developed code is confined to well-defined extension points.

5. Conclusions

The MySirt Model Execution Engine transforms a DEF model into an Executable Information System through a structured generation pipeline based on parsing, validation, model construction and transformation.

By maintaining the DEF model as the authoritative specification of the application, the Engine consistently derives application-specific artifacts while preserving a clear separation between generated components, shared runtime services and manually developed extensions.

This architectural approach makes regeneration an intrinsic property of the system, ensuring that application evolution remains driven by the declarative model while allowing controlled customization through well-defined extension points.