Empirical Evaluation of Intino as a Method for Creating and Using DSMLs

Octavio Roncal

10-2025

1 Introduction

Thank you for participating. In this session, we will introduce the *Intino* method through *Quassar*, a web platform for creating, validating, and using domain-specific modeling languages (DSMLs). We will begin with a short kata (guided demo): the instructor will create a metamodel for smart contracts, *forge* the resulting DSML, and build a sample model using this new language. From that point on, your task begins: evolve the DSML, publish a new version, and migrate the model to that version.

2 Practice Objective

To validate, from the participant's perspective, that: (i) DSML evolution can be performed in a controlled manner, (ii) version publication is straightforward, and (iii) *model migration* to the new version is clear and well-supported.

3 Session Flow

1. **Consent and background info.** Review and accept the informed consent and complete the short demographic questionnaire.
2. **Instructor's kata (10–15 min).**
   - Creation of the initial metamodel for *smart contracts*.
   - Forging of the DSML and basic verification in the editor (diagnostics, autocompletion, navigation).
   - Creation of a sample model using the new DSML.
3. **Participant task (20–30 min).** Starting from the artifacts created, you will perform the evolution, publication, and migration described in Section 4.
4. **Final questionnaires (5–10 min).** *SUS* and a short domain-specific questionnaire; open questions about the experience.

4 Participant Task: Evolution, Publication, and Migration

Starting Point

You are provided with: (i) an initial smart contracts metamodel, (ii) the corresponding forged DSML, and (iii) a sample model created by the instructor using that DSML.

Objective

Make three changes to the metamodel/DSML, publish a new version of the language, and migrate the sample model to this version, resolving any warnings/diagnostics that appear. To support this, you have access to the reference documentation of the metamodeling language, Metta, directly within the *Quassar* platform alongside the editor. Additionally, you can refer to example metamodels from other domains.

Proposed Changes

Apply the following three changes to the metamodel/DSML, reforge the language, publish a new *minor* version, and migrate the sample model.

1. Ensure the following constraints on contracts:
   - Contracts must have at least two parties.
   - The start date must be earlier than the end date.
2. A clause can be defined as recurring, indicating that it applies periodically over time. In that case, a cadence in months must be specified to determine the interval between successive repetitions.
3. The deadline can now be of two types: either absolute (e.g., "2025-12-01") or relative, by specifying a number of days from the clause's activation.

Steps to Follow

1. Go to `https://quassar.io`. If you don't have an account, register with your email. Click on *Start Building* and open the project shared by the instructor. Your work is saved automatically.
2. **Evolve the metamodel** by applying the three changes above (in any order you prefer).
3. **Versioning and forging:** Forge the DSML again, increasing the *minor* version (e.g., `1.0.0` → `1.1.0`).
4. **Model migration:** Open the sample model with the new version of the DSML and execute the migration. Address all diagnostics until the model is free of blocking errors.
5. Once finished, share the metamodel and model with the instructor through the application using the email: octavioroncal11@gmail.com.

5 Questionnaires

Once you finish, please complete the questionnaires available at the following link: `https://forms.gle/7QbyqFBbns5gUYQA7`. These forms collect your perception of the experience and the usability of the platform and the evolution/migration process.

- **SUS** (global usability).
- **Short specific questionnaire** (migration/diagnostics).
- **Open-ended questions** about encountered barriers and suggested improvements.

6 What is Being Evaluated (for Your Reference)

In this practice, we are interested in understanding *how* you evolve the language, *what support the tool provides* as you do it, and *how you perceive* the experience. We are not looking for speed records or a "perfect solution", but to observe the actual workflow and gather your perception.

- **Ease of getting started and initial effort.** We look at the approximate time it takes to get a first valid model (no syntax errors or blocking violations) from the base project, what artifacts are generated automatically, and whether you needed to add "glue" or manual steps. We will also collect your perception of the startup effort in a short specific questionnaire.
- **Language evolution and model compatibility.** We evaluate whether, after applying the proposed changes to the DSML, the sample model still works, whether the migration steps are clear, and whether the tool offers appropriate support (automatic application when possible and informative messages when manual action is required). We will collect your perception of the migration process's clarity.
- **Early detection and diagnostic quality.** We pay attention to whether issues are detected as you edit (not just at compile time), whether the messages are useful (indicating what rule is violated and where), and the *speed* at which feedback appears to maintain a smooth editing cycle. You will also be asked to assess the usefulness of the diagnostics and editor response.
- **Accessibility and autonomy.** We want to know to what extent you were able to complete the tasks without technical assistance, and whether browser-based access reduced friction from installation and setup. This will be captured in the specific questionnaire.
- **General usability.** We evaluate your overall experience using the SUS questionnaire and a brief note on the behavior of the web editor (autocompletion, navigation, diagnostics).

How the information is collected. The platform automatically logs relevant times and events (e.g., validations and migrations), and you will complete a usability questionnaire (SUS), a short specific one on migration/diagnostics, and a couple of open questions about barriers and suggestions. If something is unclear during the practice, make a note of it: we are interested in both the outcome and your experience during the process.

Thank you for your collaboration!

A Domain: Smart Contracts

A smart contract is a digital agreement between two or more parties that defines clauses, obligations, conditions, and possible penalties. It has a defined duration, is signed by parties with specific roles, and may include conditional rules that trigger certain clauses based on events, dates, or states. The goal of this domain is to design a DSL, through a meta-model, that allows the representation of smart contracts with all their essential elements.

Each contract has a unique identifier, a start date, an end date, and must involve at least two *parties*. The parties involved are characterized by a name, a unique identifier (such as a document number), and a specific role, which can be one of the following: `Buyer`, `Seller`, `Lessor`, `Lessee`, `Provider`, or `Client`.

Contracts are structured into clauses, each of which contains a title, a textual description, and optionally, a set of obligations. These obligations indicate the specific action that a party must perform—such as `Pay`, `Deliver`, `Notify`, `Transfer`, `Terminate`, or `Renew`—in addition to stating who must fulfill them and, if applicable, establishing a due date. This date can be defined either absolutely (e.g., `\2025-12-01"`) or relatively, by specifying a number of days from the activation of the clause.

Some clauses may include penalties associated with the non-fulfillment of their obligations. Each penalty consists of a monetary amount and a description and can only be defined if the clause contains at least one obligation.

Additionally, clauses may be conditional, activating only if a precondition is met. These conditions can be of two types: dependent on external events or on specific dates. If a clause is defined as conditional, it must include an expression that evaluates the condition.

Sub-clauses are also allowed, functioning as nested clauses that can include their own obligations, penalties, and conditions, following the same logic as main clauses.

Furthermore, a clause can be defined as *recurring*, indicating that it applies periodically over time. In such cases, a *monthly cadence* must be specified to determine the interval between successive repetitions, starting from the contract's start date.

Finally, the domain establishes certain semantic constraints. A contract must have at least two parties, and its start date must be earlier than its end date. A clause cannot contain incompatible actions such as `Terminate` and `Renew` simultaneously. Penalties cannot exist without associated obligations, and every recurring clause must specify a valid cadence.

B Smart Contract Examples

Example 1: Simple Purchase-Sale Contract

This contract involves two parties: Alice, with the role of `Buyer` (ID: 12345678A), and Bob, as `Seller` (ID: 87654321B). The agreement starts on September 1, 2025, and ends on October 1 of the same year.

The first clause states that Bob must deliver the product before September 10, 2025. If he fails to fulfill this obligation, a financial penalty of €100 will be applied. The second clause stipulates that Alice must make the payment to Bob before September 15, 2025.

Example 2: Conditional Service Contract

This service provision contract involves CloudCorp, acting as `Provider` (ID: C-001), and EduOrg, acting as `Client` (ID: E-101). The contract comes into effect on January 1, 2025, and runs until

December 31 of the same year.

The first clause defines an activation condition: the contract is only activated if the event `"ServiceRequestReceived"` occurs. After this event, CloudCorp must notify the activation of the service within a maximum of five days. Failure to do so results in a penalty of €250. The second clause establishes automatic renewal: EduOrg must renew the contract unless it states otherwise before December 1, 2025.

Example 3: Rental Contract with Sub-clauses

This contract involves Laura (role `Lessor`, ID: L-345) and Marco (`Lessee`, ID: M-567). The rental agreement begins on April 1, 2025, and ends on March 31, 2026.

The first clause defines a series of monthly obligations distributed in sub-clauses. In sub-clause 1.1, Marco commits to paying the rent before the 5th of each month. In sub-clause 1.2, it is stated that Laura must notify any issues before the 10th of each month.

The second clause states that, before March 15, 2026, Laura must deliver a final report on the condition of the property to Marco through a `Deliver`-type action.