Kolasu and Lionweb: an Integration Story

Alessio Stalla / Strumenta

LangDev CON 2024

Seville 17-19 October, 2024

https://langdevcon.org

Who am I?

Alessio Stalla
Language Architect at Strumenta since 2020

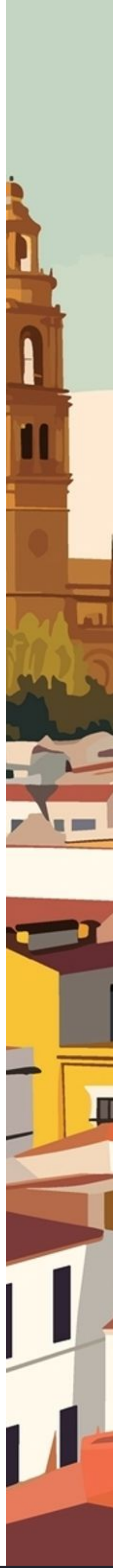
Language Engineering Enthusiast
Lisp Hobbyist

Agenda

1. Kolasu and LionWeb Recap
2. Why Integrating: Use Cases
3. Similarities and Differences
4. Discussion

U09L

ProxyHands



Kolasu

- Strumenta’s Open Source library to support language implementation **on the JVM (Kotlin)**
  - AST definition and transformations
    - AST nodes are defined as Kotlin classes
  - ANTLR Integration
  - Semantic Enrichment (symbol res., type system)
  - Interoperability → External Formats
- Part of the **StarLasu** family (multiplatform)

LionWeb

- “[A]n ecosystem of interoperable components for building **language-oriented modeling** tools on the web” https://github.com/LionWeb-io/
  - Concretely, a meta-metamodel and storage format for **models** (graphs with a primary containment hierarchy)
  - Bindings for several languages, incl. Java and TS
  - Plus a model repository to store and retrieve models (APIs + reference implementation on Node+PGSQL)

Kolasu and LionWeb

- StarLasu ASTs and LionWeb models have many traits in common (as we’ll see)
- We want **interoperability** with the LionWeb ecosystem

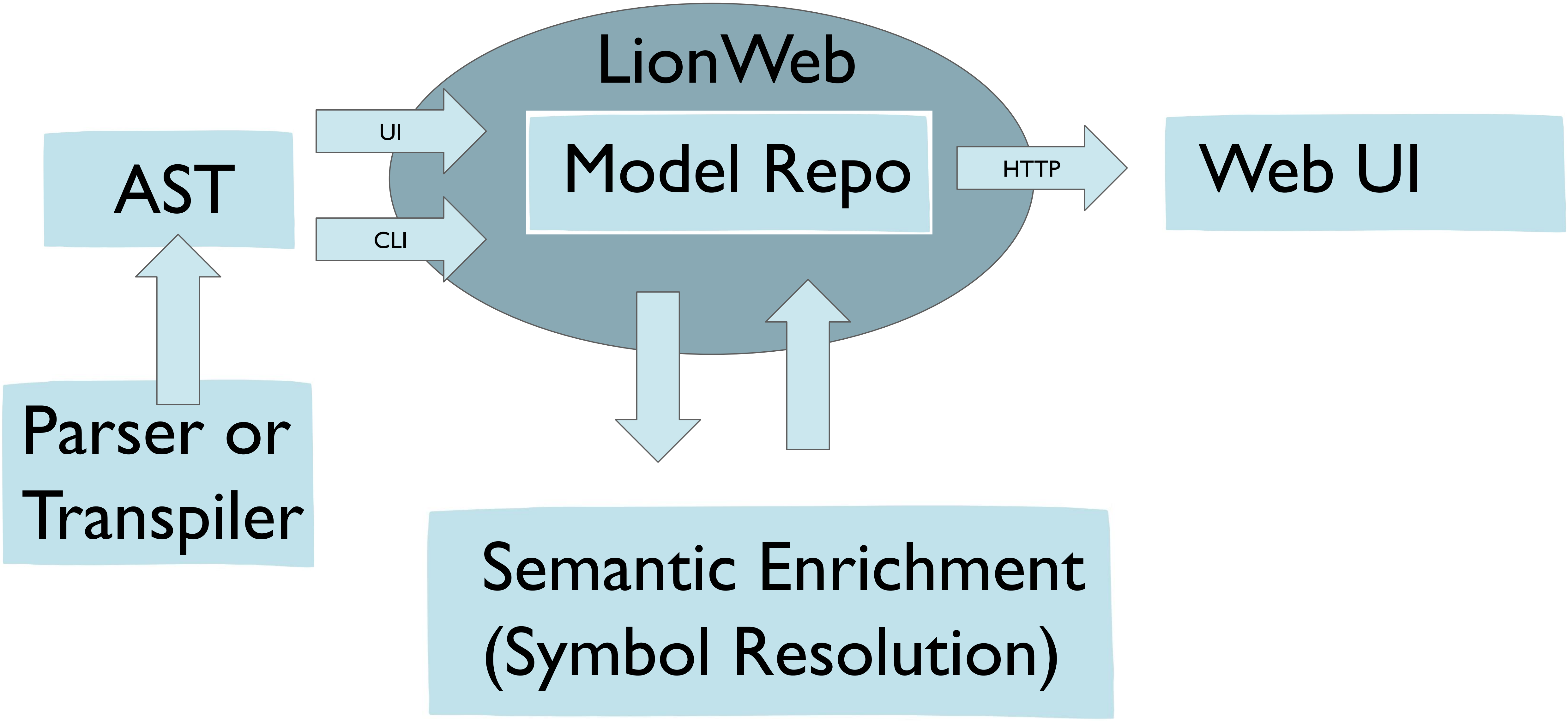
AST ≠ Model

- Early on, we decided that Kolasu AST nodes are NOT to be implementation of LionWeb-Java nodes:
  - There are some **key differences** (as we'll see)
  - Kolasu and LionWeb can **evolve independently**
- So, this is the story of how LionWeb is integrated **as an interchange format** in Kolasu

Use Cases

- To store and process StarLasu ASTs with third-party LionWeb tools (e.g., the model repository)
- To consume (some) LionWeb models as StarLasu ASTs:
  - Kolasu → LionWeb → Kolasu (backend)
  - Kolasu → LionWeb → Tylasu (frontend)
  - Third party → LionWeb → Kolasu

Kolasu+LW in Code Insight Studio



Predecessor: EMF/ECore

- Before LionWeb, Kolasu already had the capability to export to EMF/ECore (Eclipse)
- **Why LionWeb then?**
- EMF/ECore is basically Java+XML only (JS/Python+JSON implementations exists but partial, unmaintained, buggy)

Kolasu 1.5

- What we’ll discuss in the following applies to the **1.5.x** version of Kolasu, that’s currently in use at Strumenta.
- 1.6.x is mature, but hasn’t been used on real projects yet
- Maybe next year!

Trivial Mappings & Similarities

- Let’s now look at how concepts in Kolasu map to concepts in LionWeb
  - We’ll start from trivial 1-1 mappings and similar concepts
  - We’ll then discuss the most important differences

Language

- Both Kolasu and LionWeb have the concept of a *Language* which is a container of **concepts** (node classes in Kolasu), primitive types, enums, …
- In Kolasu, this used to be **implicit;** however, already with EMF/ECore interop it became necessary to explicitly list all the node classes.
  - We used to call that a *metamodel,* but we switched to *language* to better match LionWeb terminology.

Concept (1)

- A **Concept** represents the definition of a class of nodes. It has a **name** and a number of **features.**
- In Kolasu 1.5, this is **implicit:** we use Kotlin **reflection** to derive the structure of a node – which properties are **attributes, containments, references,** or **internal/computed attributes.**
- Kolasu looks at the **type** and **annotations** of each property to determine its role.

Concept (2)

- A **Concept** in LW can have **ancestors,** IOW, LionWeb supports traditional OO **inheritance.**
- Kolasu naturally supports inheritance because it uses Kotlin classes as concepts.
- We can conclude that, despite the fact that Kolasu 1.5 doesn’t have an explicit representation of a **concept,** it has all the capabilities required to derive one.

Concepts, Languages, and Packages

- Roughly speaking, **concepts are classes** in Kolasu
- Kotlin supports **packages** to organize classes
- However, **packages** and **languages** are different:
    - A **language** allows to **list** all its elements
    - A **package** does not (easily)
    - A **language** is a higher-level concept, and could include nodes from different packages.
    - Packages are just a way to **organize source code** and **avoid name clashes** between different codebases.

Features

- In LionWeb, **features** are characteristics of a concept:
  - **Attributes** of primitive types (can’t be multiple)
  - **Containments** of other nodes (possibly multiple)
  - **References** to other nodes in the graph (single or multi)
- In Kolasu, we adopted the same terminology and similar constraints (e.g. no lists of primitive types). However, in 1.5, we still refer to features as **node properties** taking the terminology from Kotlin (and JavaBeans).

Primitive Types

- Kolasu doesn’t restrict primitive types: everything that’s not a Node is a primitive type.
    - ⇒any property whose type is not Node or reference is an attribute.
- However, if we want to export values of a primitive type that is not one of the LionWeb built-in ones, we have to provide a serializer (and deserializer on the other end).

Containments

- A containment is just a property whose type is a subtype of Node, or a list whose elements are of a subtype of Node
- The contained node has a reference back to its parent.
- In Kolasu 1.5, the parent isn’t always managed automatically: some operations set it for you, but you can also manually attach a node without setting its parent.
- In Kolasu 1.6, parent tracking is fully automated.

References

- Kolasu has the concept of *ReferenceByName,* an object that has:
  - a *name* used as a key to resolve a reference
  - a pointer to the referred object
  - and other bookkeeping information that we'll gloss over
- References are resolved during **semantic enrichment**
- They're just references in memory
- How do we map them to LionWeb references?

Storage Model and Node IDs

- Kolasu: all **AST nodes are in memory** (conceptually)
- LionWeb: **not all nodes** are loaded in memory, or from the same file, database, repository, etc.
- Kolasu: AST nodes are usually created as the result of **parsing** source code, or **transforming** another AST (derived models)
- LionWeb: nodes may be authored using a **projectional/structure editor** (e.g. Freon, MPS, …) ⇒ they may have an **identity** of their own

Storage Model and Node IDs

- As a consequence:
  - **All LionWeb nodes have an ID**
  - Kolasu AST nodes **don’t** have an ID
- ⇒ when converting Kolasu AST nodes into LionWeb nodes, **we have to provide an ID**
- ⇒ when converting Kolasu references into LionWeb ones, we need to go through the node ID (which also allows to store the target node elsewhere).

Node ID Strategies (1)

- We could consider various **constraints** when choosing a strategy to assign IDs
  - Should a node ID remain **constant across runs** of the application? E.g. if we **parse the same file twice** in two different executions, should we maintain the same IDs?
  - Should a node ID remain **constant across transformations** of the AST? For example, if we **move a statement** inside a method, should it keep the same ID?

Node ID Strategies (2)

- Some possibilities:
  - **Compute** the ID using some defining **attributes** of the node **(semantic ID)**. E.g., the ID of a Java class node is its **package+name** (with maybe **Maven coordinates**)
  - **Compute** the ID using the **path** from a well-defined ancestor node, for example, in a Java method, *the third statement in the first for statement*
  - **Randomly assign** the ID (e.g. with a UUID)

Node ID Strategies

- In Kolasu we have a *NodeIdProvider* interface with several built-in implementations to provide the node ID strategy
- The default strategy is the most flexible one:
  - If we know how to compute the *semantic ID* of a node, do it (we have an IIN or Independent ID Node)
  - Otherwise, compute the ID combining the parent node’s ID and the node’s path in the parent
  - This requires that the **root** of the AST is an IIN (it could be a synthetic node representing the parsed file with its path or checksum as the ID)

Partitions

- Lionweb divides models in **partitions**
- Kolasu doesn’t have such a concept
- Presently, we only deal with partitions when using the model repository API, and we don’t represent them as AST nodes

Objects that are not Nodes

- Kolasu has additional types of support objects that are not AST nodes.
- When serializing these types of Kolasu objects to LionWeb, we represent some of them as primitive types, and some others as special nodes.
- Let’s look at them.

Objects that are not Nodes

- ***Point*** and ***Position*** represent line and column info (to track a node's position in the source code).
  - Structured data ⇒ nodes
  - However, **nodes aren't cheap,** and every AST node has a position which is made of 2 points → 3 extra nodes
  - So, we represent these as **custom primitive types**

Objects that are not Nodes

- ***Source*** represents where the node comes from (e.g. a file with a certain path)
  - we omit it when serializing into LionWeb nodes
- ***Issue*** represents some issue in parsing (syntactic or semantic) or just an information message
  - Represented as a LionWeb node

Objects that are not Nodes

- ***Token*** is a portion of the input text with an associated type, can be used for syntax coloring
  - A single *Token* is an element of a list
  - Making them into nodes is not cheap
  - However, attributes cannot be multiple
  - ⇒ we “cheat” and create a primitive type representing a “list of tokens” as a single value
- ***ParsingResult*** is an AST + issues + tokens
  - Just a LionWeb node with attributes + contained nodes

Other Advanced Features

- Kolasu-native client for the LW Model Repository
  - Based on LW-native client in LW-Kotlin
- Proxy Nodes
- CLI commands and Gradle tasks for:
  - Generating a LW Language from a Kolasu AST
  - Generating Kolasu AST classes from a LW Language

Wrapping It Up

- Kolasu and LionWeb have enough similarities so it’s sensible to use LionWeb as an external format for Kolasu ASTs
- Some differences exist especially around node IDs so some extra complexity is needed
- Nonetheless we could use Kolasu and LionWeb together successfully in our Code Insight Studio tool

U09L

ProxyHands

Q&A

Thanks

“ Thank you!

https://twitter.com/strumenta

https://www.linkedin.com/company/strumenta

STRUMENTA
COMMUNITY

https://strumenta.community/

https://strumenta.com/

STRU
MEN
TA