LangDev Meeting 2018, Amsterdam

PORTING THE WHOLE LWB TO SWIFT/IOS

SWIFT GENERATORS

Riccardo Solmi, Whole Factory, Italy

THE WHOLE PLATFORM IS AT THE STATE OF THE ART

- DLSs covering almost every aspect of language definition
- Visual notations designed for gesture based interactions
- DSLs for evolution, testing and deployment
  - multiple versions, instance migration, software product lines

. . . BUT WE ARE UNDERGOING A SLOW EVOLUTION

- The Whole Platform is not where we want it to be
- Implementation is far behind our vision
- New features are struggling to become pervasive

TOO MUCH INNOVATION BRAKES: UNDERLYING TECHNOLOGIES

- Java is no longer Write once and Run Everywhere
  - (iOS, Windows 10 UWP)
- GEF 3 is no longer developed and is outdated
  - (Draw 2D, poor Multitouch, non composable UI, Java)
- Eclipse is too much big and complex for too little
  - (Classic IDE, Java)

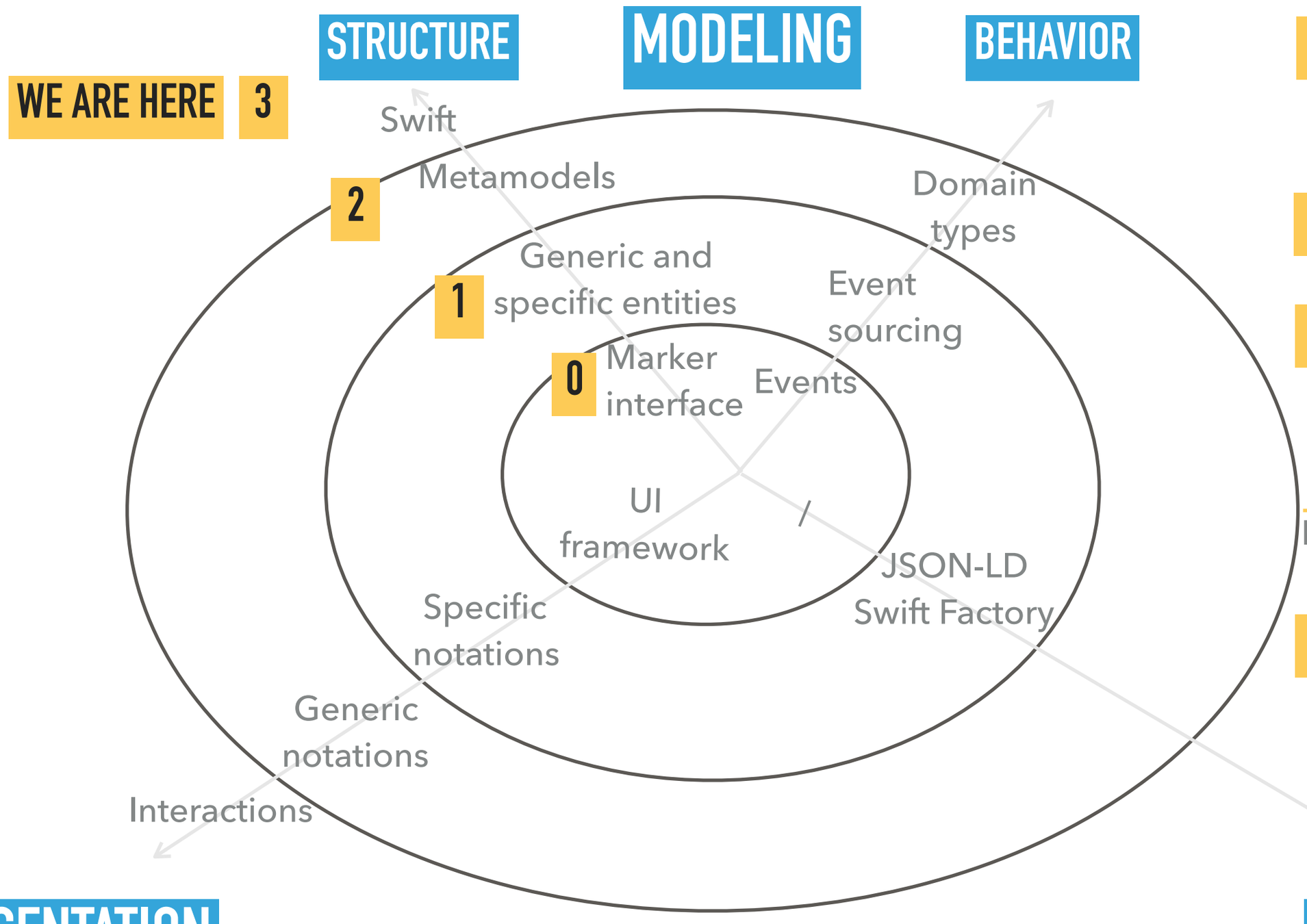
TOO MUCH INNOVATION BRAKES: WHOLE FRAMEWORK

- Duality: framework level <-> domain level
  - Duplication, encoding, constrained domain innovation
- Framework APIs in well-established but wrong places
  - Modeling, events, reflection, and typing
- Framework evolution is inherently slower than domain level evolution

SO WE NEED TO START OVER AGAIN?

- No, domain level code and generated code account for 98% of the code base
- We decided to reuse the domain level and to redesign and rewrite the framework level
- Swift + iOS are the technologies chosen for the porting
- Existing Eclipse based workbench has been used to accelerate the bootstrapping process

OVERVIEW OF THE PORTING PROCESS: FIRST ITERATIONS



0 Learning
Experimenting
Prototyping UX

1 Generating
model structure

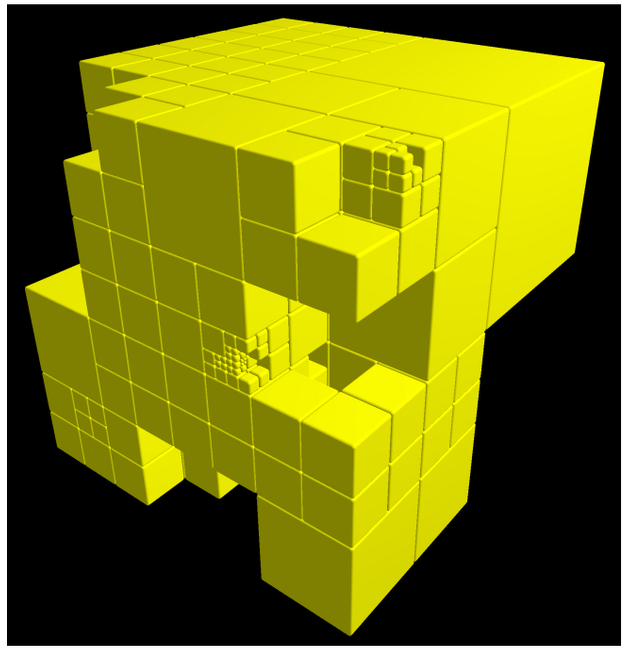
2 Domain typing
using existing
metamodels

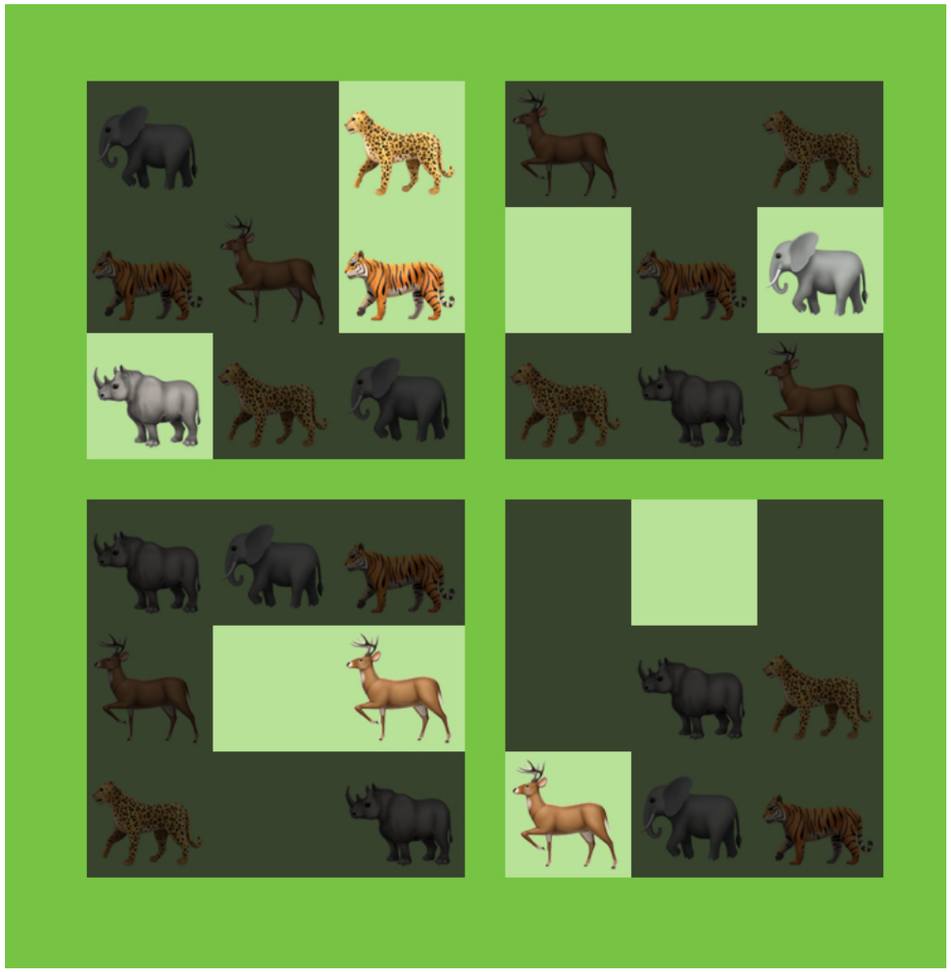
Structural bootstrap
Behavioral bootstrap

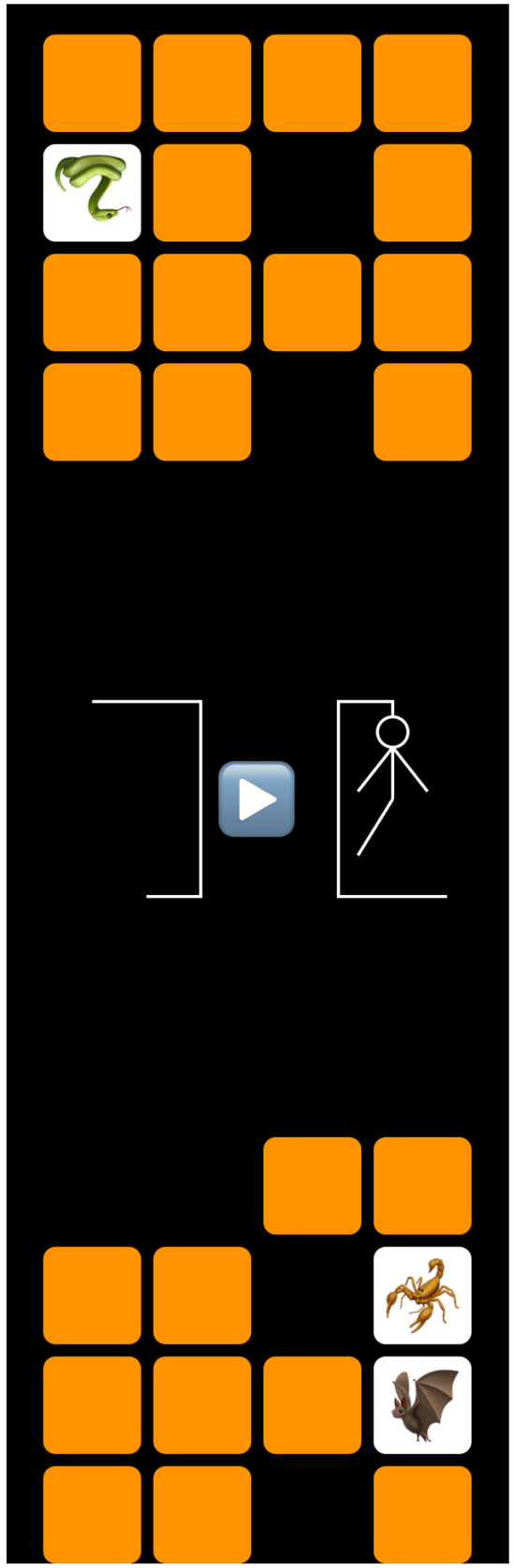
3 Supporting
translational
semantics for
Swift

ITERATION 0

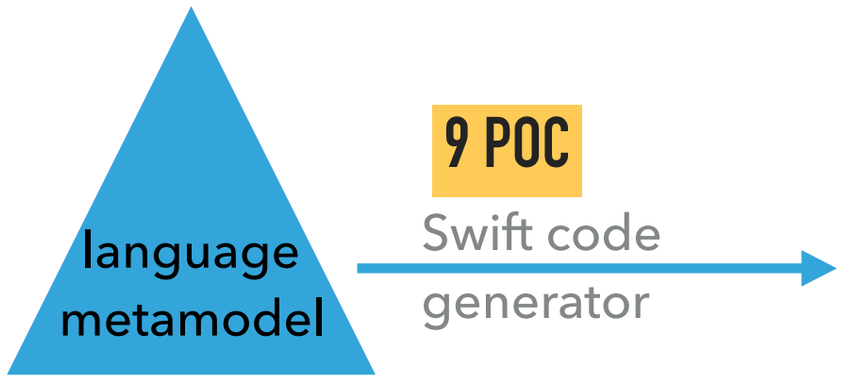
FIRST ITERATION PROTOTYPES: SIMPLE GAMES







GENERATING LANGUAGE IMPLEMENTATION (STRUCTURE)



	JSON	Models	SwiftSyntax	SwiftAST (Swift*)
Swift implementation (specific APIs)	122 LOC	662 LOC	6104 LOC	4550 LOC 1762 LOC
Swift Factory serialization	78 LOC	316 LOC	3465 LOC	2510 LOC 967 LOC
Generation time	2 S	3.5 S	100 S	60 S 12 S

* Metamodel hand written starting from the official grammar

ITERATION 1

MODELS ON IOS

- specific notation

URI	whole:org.whole.lang.json:JSONModel
Namespace	org.whole.lang.json
Model Name	JSON
Version	Resolver
Foreign Types	SupertypesOf
types	Value
foreignType	AnyType

Supertypes	Entity	Structure		
_	Value	Modifiers	Feature	Type
Value	Object	_		
_	Pair	Modifiers	Feature	Type
		_	name	Name
		_	value	Value
_	Name	String		
Value	Array	_		
Value	String	String		
Value	Decimal	double		
Value	Int	long		
Value	Bool	boolean		
Value	Null	Modifiers	Feature	Type

URI	whole:org.whole.lang.swift:SwiftModel
Namespace	org.whole.lang.swift
Model Name	Swift
Version	Resolver
Foreign Types	_

Supertypes	Entity	Structure		
	CodeBlock	_		
_	Statement	Modifiers	Feature	Type
Statement	Declaration	Modifiers	Feature	Type
Declaration	ProtocolDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	inheritedProtocols	Protocols
		_	declarations	ProtocolMemberDeclarations
_	ProtocolMemberDeclarations	_		
_	ProtocolMemberDeclaration	Modifiers	Feature	Type
ProtocolMemberDeclaration	ProtocolPropertyDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	type	TypeAnnotation
		optional	hasSetter	BooleanLiteral
ProtocolMemberDeclaration	ProtocolMethodDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	genericParameters	GenericParameters
		_	signature	FunctionSignature
Declaration ClassMemberDeclaration EnumerationMemberDeclaration	MemberDeclaration	Modifiers	Feature	Type
MemberDeclaration	EnumerationDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	genericParameters	GenericParameters
		optional	adoptedProtocols	Protocols
		_	declarations	EnumerationMemberDeclarations
_	EnumerationMemberDeclarations	_		
_	EnumerationMemberDeclaration	Modifiers	Feature	Type
MemberDeclaration	StructureDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	genericParameters	GenericParameters
		optional	adoptedProtocols	Protocols
		_	declarations	StructureMemberDeclarations
_	StructureMemberDeclarations	_		
MemberDeclaration	ClassDeclaration	Modifiers	Feature	Type
		_	name	Name
		optional	genericParameters	GenericParameters
		optional	superclass	TypeIdentifier
		optional	adoptedProtocols	Protocols
		_	declarations	ClassMemberDeclarations
_	ClassMemberDeclarations	_		
_	ClassMemberDeclaration	Modifiers	Feature	Type
MemberDeclaration	FunctionDeclaration	Modifiers	Feature	Type
		optional	modifiers	FunctionModifiers
		_	name	Name
		optional	genericParameters	GenericParameters
		_	signature	FunctionSignature
		_	body	CodeBlock
_	FunctionSignature	Modifiers	Feature	Type
		optional	parameters	Parameters
		optional	throwsModifier	ThrowsModifier
		optional	returnType	Type
	ThrowsModifier	THROWS		

ITERATION 1

PERSISTENCE

- To bootstrap the platform by loading the metamodels
- XML Builder (too much complex for the bootstrap)
- JSON-LD (unordered -> cannot bootstrap metamodels)
- Swift Factory

ITERATION 1

SWIFT FACTORY PERSISTENCE

- No parser needed
- Suitable for embedding domain templates in Swift code

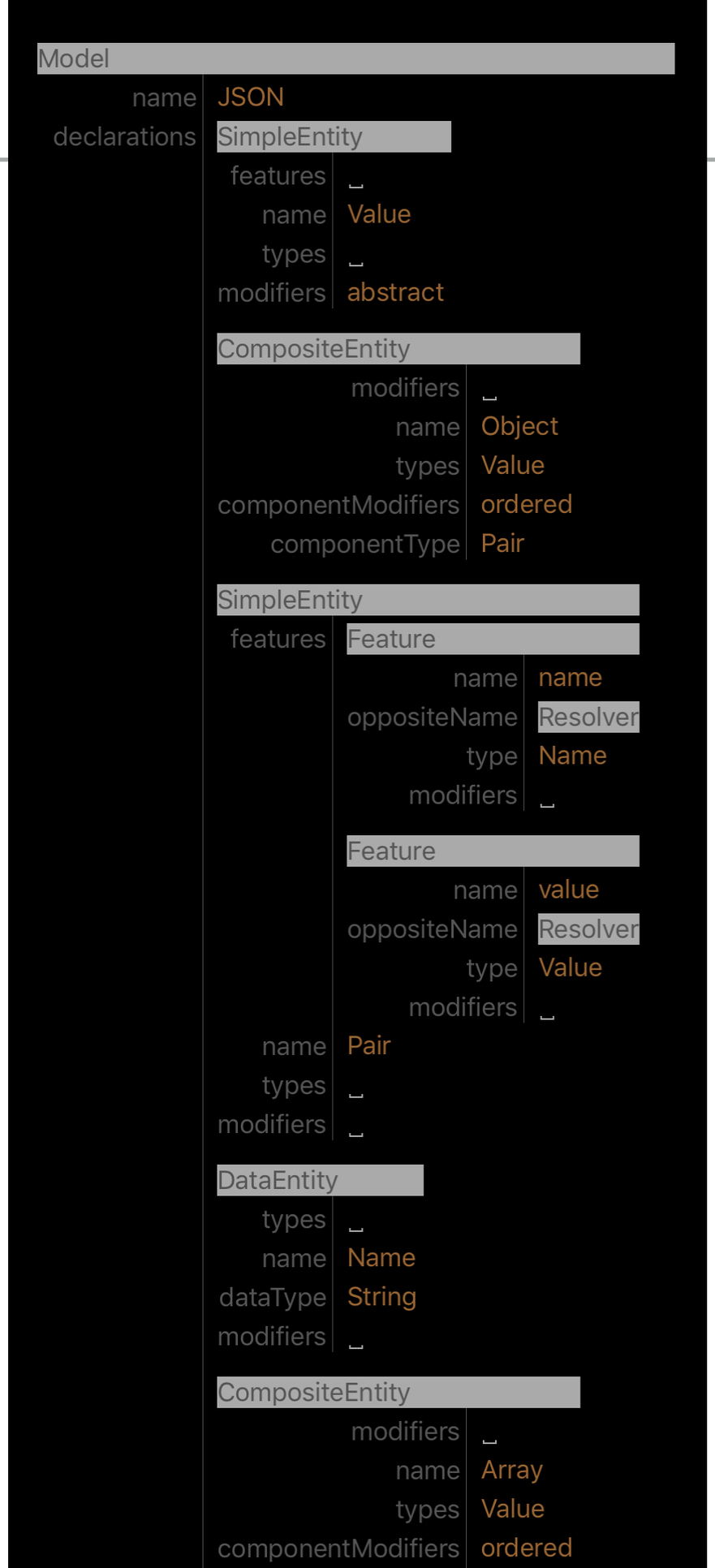
```
import Foundation

public func createSwiftASTModel(context: MoldingContext) -> Models_Model {
    let modelsEF = ModelsFactory(context: context)
    let commonsEF = CommonsFactory(context: context)
    return modelsEF.Model(
        name: modelsEF.SimpleName("SwiftAST"),
        typeRelations: commonsEF.Resolver(),
        declarations: modelsEF.ModelDeclarations(
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Syntax"),
                types: modelsEF.Types(),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Decl"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Expr"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Pattern"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Stmt"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(modelsEF.EntityModifier(.abstract)),
                name: modelsEF.SimpleName("Type"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features()),
            modelsEF.SimpleEntity(
                modifiers: modelsEF.EntityModifiers(),
                name: modelsEF.SimpleName("Token"),
                types: modelsEF.Types(modelsEF.SimpleName("Syntax")),
                features: modelsEF.Features(
                    modelsEF.Feature(
                        modifiers: modelsEF.FeatureModifiers(),
                        type: modelsEF.SimpleName("TokenKind"),
                        name: modelsEF.SimpleName("kind"),
                        oppositeName: commonsEF.Resolver()),
                    modelsEF.Feature(
                        modifiers: modelsEF.FeatureModifiers(modelsEF.FeatureModifier(.option
                        type: modelsEF.SimpleName("Text"),
```

ITERATION 2

GENERIC NOTATIONS

- “Table” variant
- Dark theme



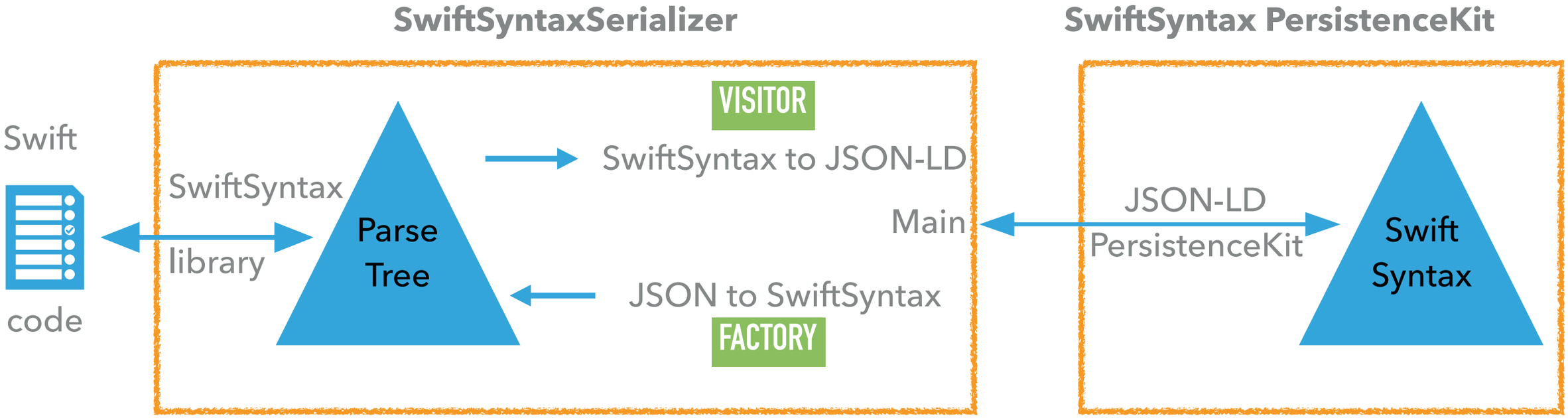
ITERATION 3

SWIFT INTEGRATION THROUGH REVERSE ENGINEERING

- Add concrete syntax level Swift persistence to Eclipse backed by official Swift parser
  - Using a fully automated generative process
  - Suitable for source editing scenarios
- Add AST level Swift persistence to Eclipse
  - Chained to, and initially derived by the syntax level
  - Refined to better support generative scenarios

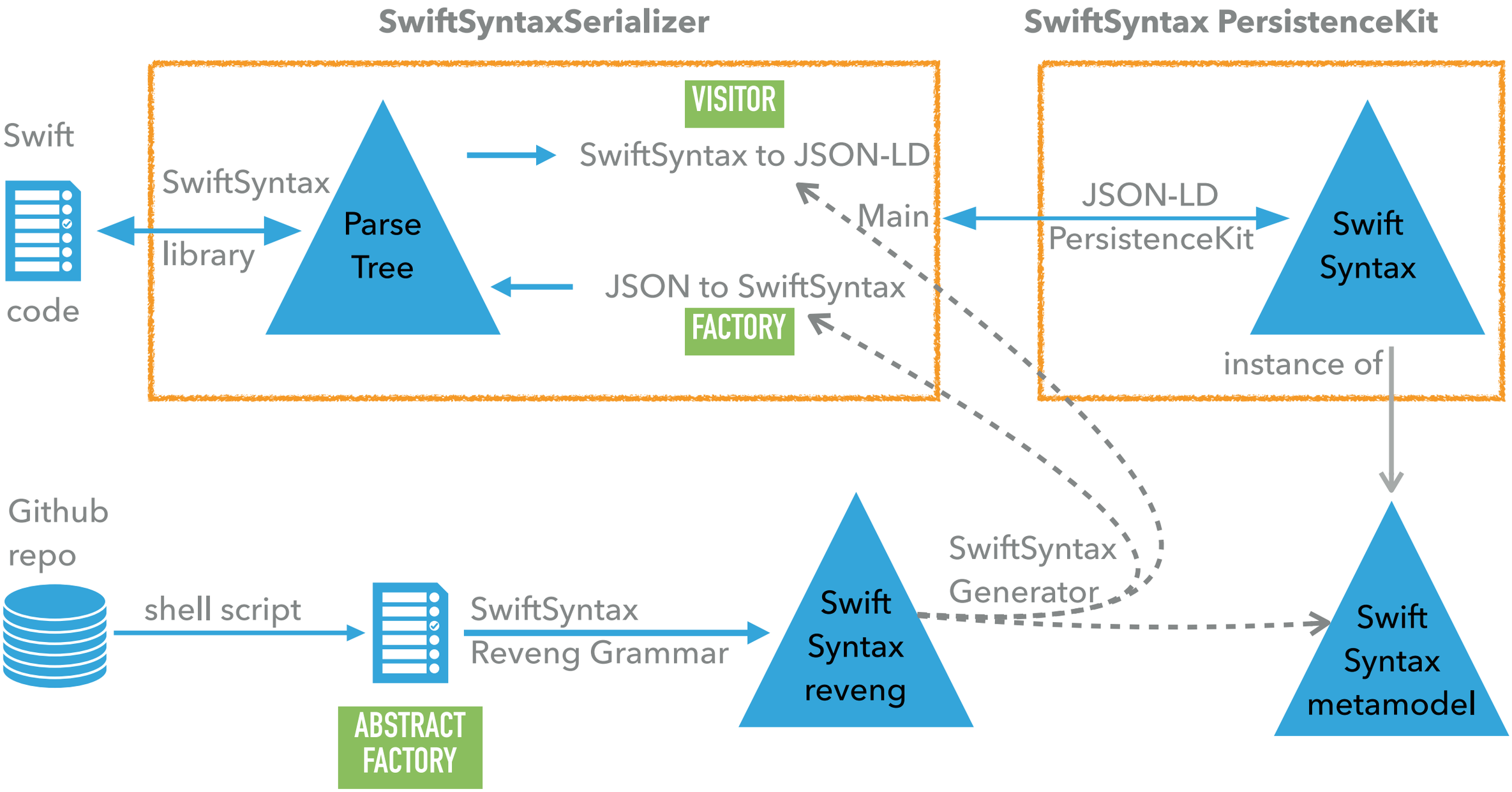
ITERATION 3

SWIFTSYNTAX PERSISTENCE

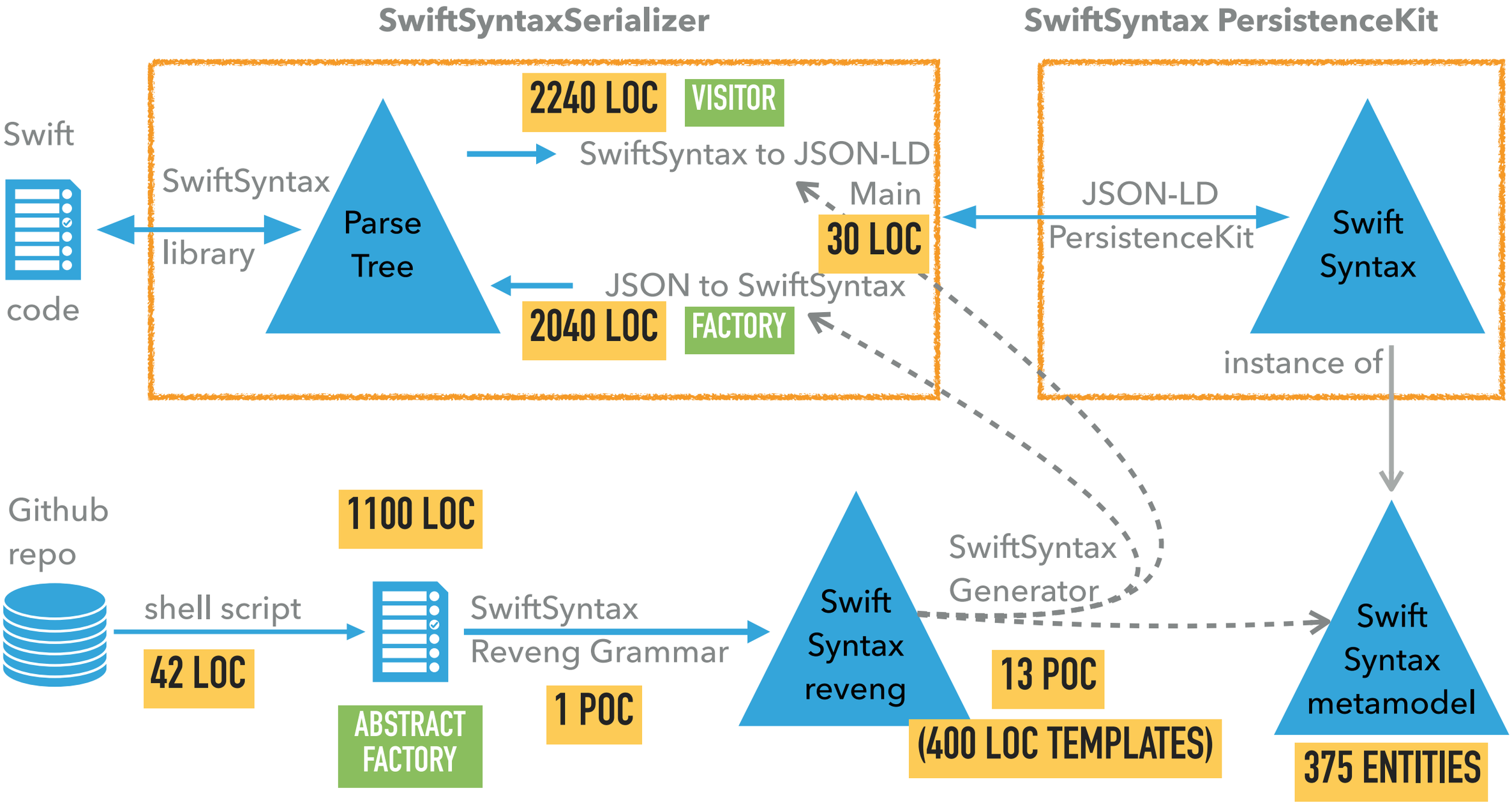


ITERATION 3

SWIFTSYNTAX PERSISTENCE



SWIFTSYNTAX PERSISTENCE

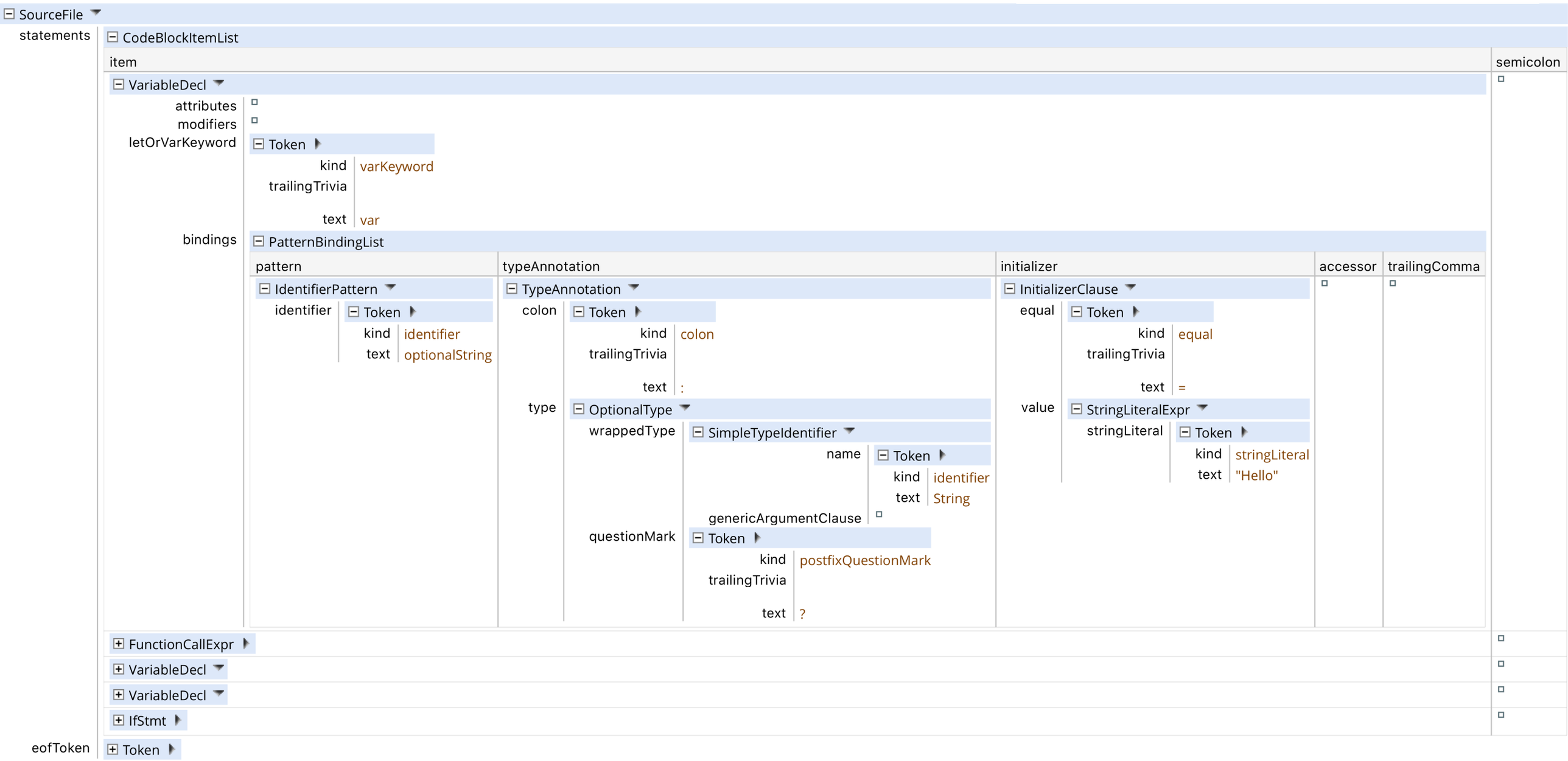


ITERATION 3

SWIFT SYNTAX EXAMPLE

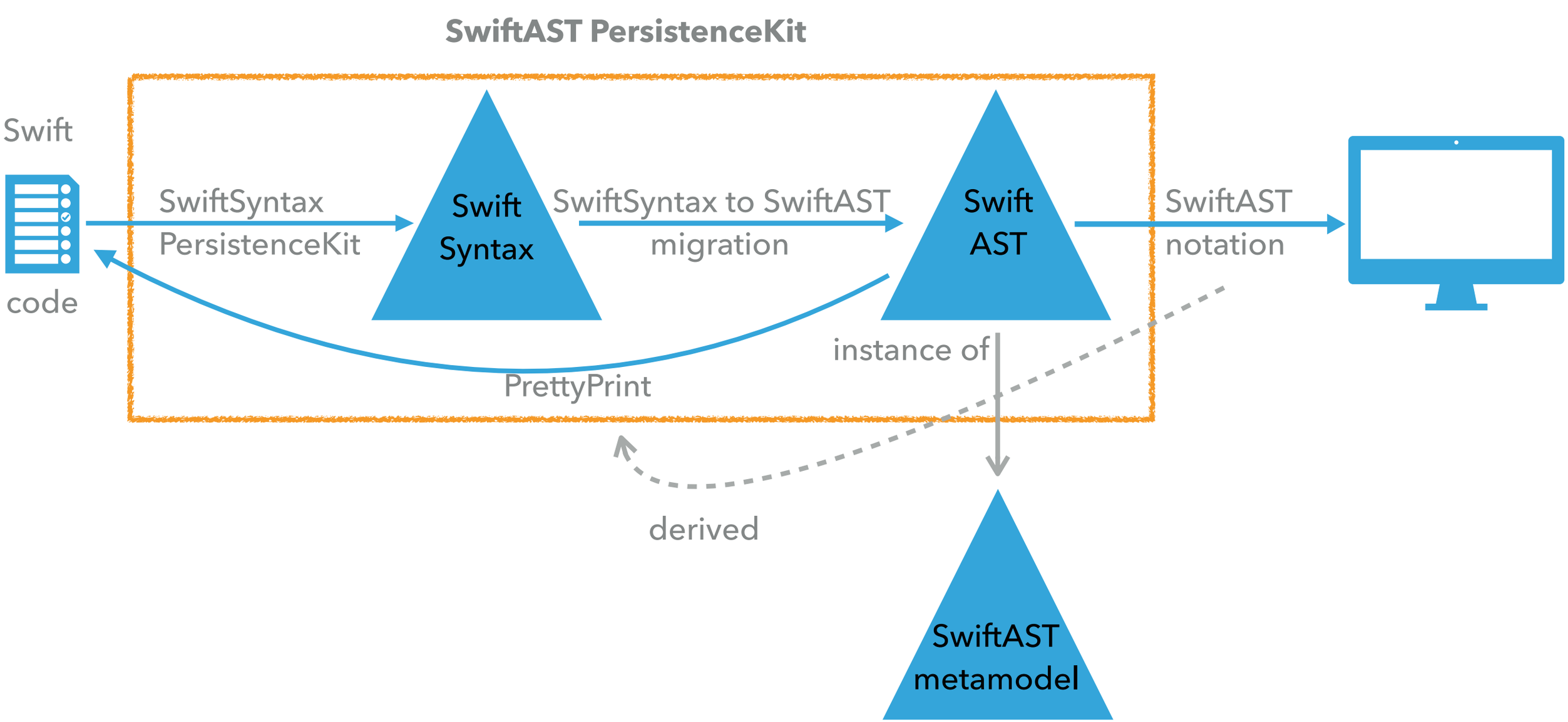
```
var optionalString: String? = "Hello"
print(optionalString == nil)

var optionalName: String? = "John Appleseed"
var greeting = "Hello!"
if let name = optionalName {
    greeting = "Hello, \(name)"
}
```



ITERATION 3

SWIFTAST PERSISTENCE



ITERATION 3

SWIFT AST EXAMPLE

```
var optionalString: String? = "Hello"
print(optionalString == nil)

var optionalName: String? = "John Appleseed"
var greeting = "Hello!"
if let name = optionalName {
    greeting = "Hello, \(name)"
}
```

```
▫ ▫ var optionalString : String ▫ ? = "Hello" ▫

print ▫ ▫ : optionalString ▫ operatorToken : BinaryOperatorExpr {
                                                 Token {
                                                 kind : spacedBinaryOperator
                                                 text : ==
                                                 }
                                             } nil ▫

▫ ▫ var optionalName : String ▫ ? = "John Appleseed" ▫

▫ ▫ var greeting : ▫ = "Hello!" ▫

if ▫ let name : ▫ = optionalName ▫ {

                                    Hello,
    greeting ▫ AssignmentExpr \( name ▫ )


} else {
    ▫
}
```

ITERATION 3

SWIFT AST EXAMPLE (2)

```
import □ □ Foundation
□ public func createModelsModel □ ( □ context □ : MoldingContext □ □ = □ ) □ -> Models_Model □ □ {
  □ □ let modelsEF : □ = ModelsFactory □ ( context : context □ ) □ □
  □ □ let commonsEF : □ = CommonsFactory □ ( context : context □ ) □ □
    return modelsEF □ . Model □
      ( name : modelsEF □ . SimpleName □ ( □ : "Models" ) □
        typeRelations : modelsEF □ . TypeRelations □ ( □ ) □
        declarations : modelsEF □ . ModelDeclarations □
          [ □ : modelsEF □ . SimpleEntity □
            ( modifiers : modelsEF □ . EntityModifiers □ ( □ ) □
              name : modelsEF □ . SimpleName □ ( □ : "Model" ) □
              types : modelsEF □ . Types □ ( □ ) □
              features : modelsEF □ . Features □
                [ □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "SimpleName" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "name" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                  □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ : modelsEF □ . FeatureModifier □ ( □ : . optional □ ) □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "TypeRelations" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "typeRelations" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                  □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "ModelDeclarations" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "declarations" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                  □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "Namespace" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "namespace" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                  □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ : modelsEF □ . FeatureModifier □ ( □ : . optional □ ) □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "Version" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "version" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                  □ : modelsEF □ . Feature □
                  ( modifiers : modelsEF □ . FeatureModifiers □ ( □ ) □
                    type : modelsEF □ . SimpleName □ ( □ : "URI" ) □
                    name : modelsEF □ . SimpleName □ ( □ : "uri" ) □
                    oppositeName : commonsEF □ . Resolver □ ( □ ) □ ) □
                ] □
              □ ) □
          ] □
      □ ) □
  }
```

```
import Foundation

public func createModelsModel(context: MoldingContext) -> Models_Model {
  let modelsEF = ModelsFactory(context: context)
  let commonsEF = CommonsFactory(context: context)
  return modelsEF.Model(
    name: modelsEF.SimpleName("Models"),
    typeRelations: modelsEF.TypeRelations(),
    declarations: modelsEF.ModelDeclarations(
      modelsEF.SimpleEntity(
        modifiers: modelsEF.EntityModifiers(),
        name: modelsEF.SimpleName("Model"),
        types: modelsEF.Types(),
        features: modelsEF.Features(
          modelsEF.Feature(
```