Ballerina Swan Lake

Compiler Construction Insights from the Ballerina Language

September 2024

Lecture Outline

- Evolution of Ballerina Compiler
- Current structure of Ballerina Compiler
- Staged approach and intermediate representations
- JVM backend experience
    - Demo
- LLVM backend experience

WSO2's history in languages

WSO2 enables thousands of enterprises, including hundreds of the world's largest corporations, top universities, and governments, to drive their digital transformation journeys—executing more than 18 trillion transactions and managing more than 500 million identities annually.

Apache Synapse

XML based DSL for specifying service mediation.
Community project, heavily contributed by WSO2.

Integration Studio

Graphical editor for Apache Synapse

Jaggery Runtime

JS runtime written in Java.

Evolution of

Ballerina

Swan Lake

Compiler

- Started out as Synapse replacement language back in late 2016. Inspired by sequence diagrams and graphical editing.
- Initial implementation as AST interpreted language (2017)
- Internal vm (BVM) with internal ByteCode (late 2017)
- Backend/frontend separation via BIR. JVM bytecode as the backend (late 2018).
- Swan Lake version GA release in , with major improvements and extensive set of standard libraries and connectors (early 2022).
- Continuous updates to Swan Lake version. Currently on update 10.

Features of Ballerina

Data oriented

Type-safe, declarative processing of JSON, XML, and tabular data with language-integrated queries.

```
type User record { int id; string name; };

User manu = { id: 92874, name: "manuranga" }
```

Concurrent

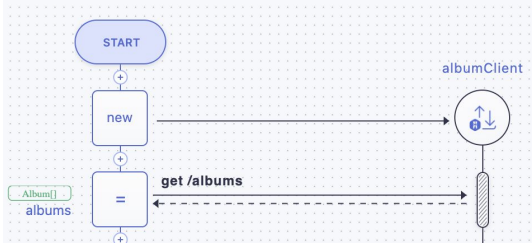
Easy and efficient concurrency with sequence diagrams and language-managed threads without the complexity of asynchronous functions.

```
http:Client hello = check new ("http://hello.com");
MyGreeting greeting = check hello->get("/world");
```

Also see: start, wait and workers

Graphical

Programs have both a textual syntax and an equivalent graphical form based on sequence diagrams.



Features of Ballerina

Flexibly typed

Uses structural types with support for openness for static typing within a program and for describing service interfaces.

Reliable, maintainable

Explicit error handling, static types, and concurrency safety, combined with a familiar, readable syntax make programs reliable and maintainable.

Cloud native

Network primitives in the language make it simpler to write services and run them in the cloud.

```
type Customer record {|
    int id;
    string name;
    int account;
|};
...
Customer customer = { ... };
User user = customer;
addUser(user);
```

```
> bal build
Compiling source
        example/greeter:0.1.0

Generating executable

Generating artifacts...

        @kubernetes:Service                 - complete 1/1
        @kubernetes:Deployment          - complete 1/1
        @kubernetes:HPA                     - complete 1/1
        @kubernetes:Docker                  - complete 2/2
```

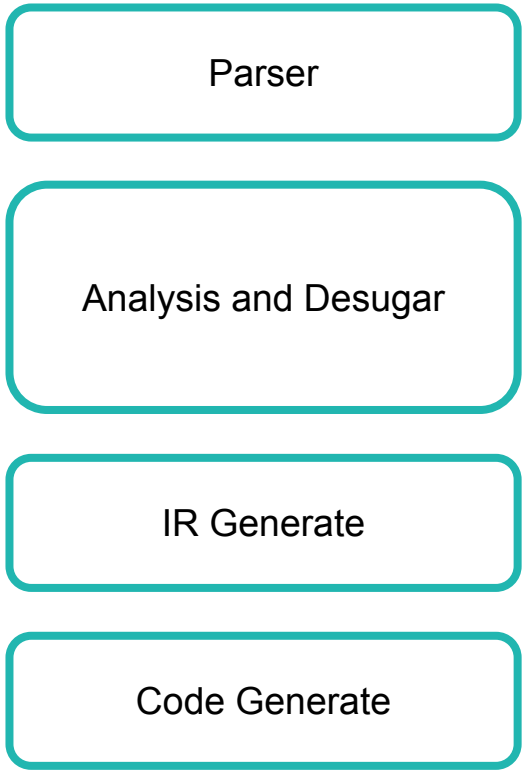

6

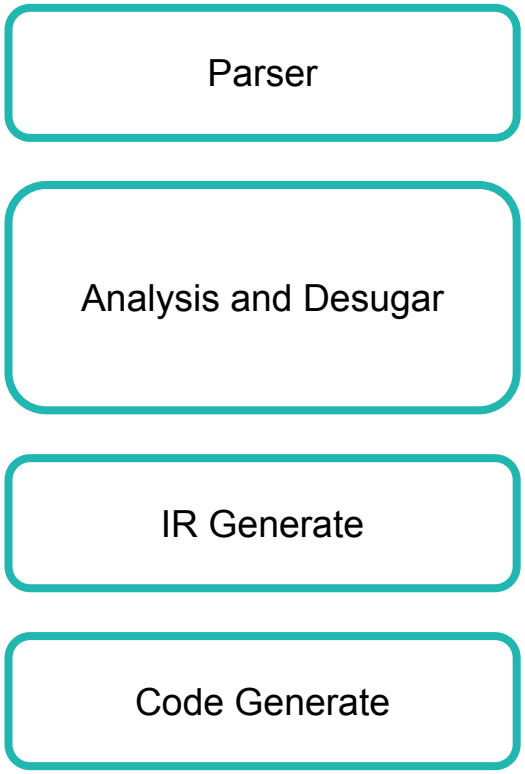
Structure of

Ballerina

Swan Lake

Compiler





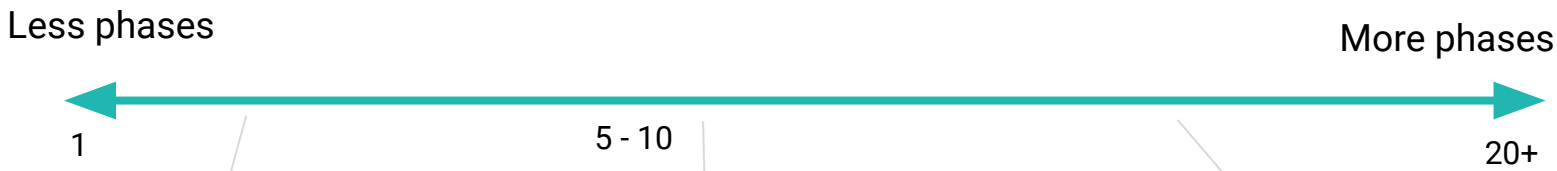
We used to have ANTLR, now we have a custom parser

Create symbols, Type check, Run plugins
Lower lambdas and other high level control flow
See CompilerPhaseRunner Class of ballerina-lang repo

Create BIR. Conditionals get lowered to GOTOs

Create JVM Bytecode. Create concurrency yield points.
Generate classes for values.

Number of compiler phases



- Less moving parts, simpler

  Currently used in nBallerina

- More people can work code base independently

  Used in Swan Lake

- Micro pass approach
- Each pass is logical simple

  Used in LLVM optimizer.

Intermediate representation

Q: How are compilers phases are connected? A: IR

Why IR (instead graph)

- Easy to debug due to serialization
- Can verify
- Share backend

Styles of IR

- Stack based vs Register based
- Flat vs structured
- SSA Register vs Mutable Register

Sample of new Ballerina IR

```
(defns
("main" (public) (function (() ()) (file "hello") (loc 3 16)
        (registers
        (r0 tmp list)
        (r1 tmp ()))
        (blocks
        (b0 (no-panic)
                (list-construct r0 "hello")
                (call (module-get ("ballerina" "io") "println") r1 r0)
                (ret ()))))))
```

IR Styles : Stack machine vs Register base

```
a = b + 20;
```

Stack machine

```
iload %1
iload 20
iadd
```

- May produce smaller IR.
- Used by JVM and WebAssembly

We didn't consider this option due to the added complexity.

Register base (mutable)

```
iload 20 %2
iadd %1 %2 %0
```

- More closer to source language.

We use this format in Ballerina (BVM, jBallerina IR, nBallerina IR)

Register base (SSA)

```
%a = add %b 20
```

- Better for analysis and optimization.
- Need phi nodes

We didn't pick because not a good input format for JVM or for LLVM (surprisingly, due to debug info)

IR Styles : Flat vs Structured

Flat IR

```
loopHead:
      local.get $i
      i32.const 10
      i32.gt_s loopEnd
      goto loopHead
loopEnd:
```

- Most popular format
- Can result in non-reducible loops

Structured IR

```
(loop $my_loop
      local.get $i
      i32.const 10
      i32.lt_s
      br_if $my_loop)
```

- Used by WebAssembly
- No GOTOs

Ballerina native backend

We use LLVM library
We really on alloca and mem2reg

Tools to get started with LLVM Code gen

- Godbolt website
- Show phi nodes in action

Ballerina JVM backend

We use ASM library.
Using visitor pattern Ballerina IR and generate JVM IR

Tools to get started with JVM Code gen

- javap command. What java classes lookalike
- ASM library and ASMPlugin in Intellij

Demo

- Write a compiler backend
- AST -> Java Class
- Extend the demo to win gifts