

Ballerina

A modern programming
language focused on integration

Sanjiva Weerawarana, Ph.D.
Founder, Chairman & Chief Architect;
WSO2

James Clark
Independent

Joint work with Sameera Jayasoma, WSO2; Hasitha Aravinda, WSO2;
Dr. Srinath Perera, WSO2; and Prof. Dr. Frank Leymann; Univ. of Stuttgart

Agenda

- Motivation
- Type system
- Network awareness, security & resiliency
- More than the language
- Implementation
- Future work

Why yet another language?

- Digital takeover
 - Everything is a network service
 - Produce, not just consume network services
 - Every business has to become a software company
- Integration technology
 - ESBs and more
- Existing mainstream languages
 - Distribution, network data/security are foreign
 - Frameworks galore

Design inspirations

- London 2012 Olympics opening
- Sequence diagrams
- Many existing languages including Java, Go, C, C++, Rust, Haskell, Kotlin, Dart, Typescript, Javascript, F#, Swift, RelaxNG



Source:

https://stillmed.olympic.org/Global/Images/News/2012-07/27/opening_country.jpg

Design principles

- Code, not config, with both text and graphic syntaxes
- Do not try to hide the network
- First class support for network data types
- No room for best practices or “findbugs”
- Make programs secure by default
- Make programs network resilient by default
- No dogmatic programming paradigm
- Mainstream concepts & abstractions, not research

Unusual aspects of Ballerina

- First-class nature of application level network abstractions
- Structural type system with network friendly types and unions
- Sequence diagram based parallelism model
- Language extensibility with environment binding and compilation extensibility

Hello, World from Ballerina

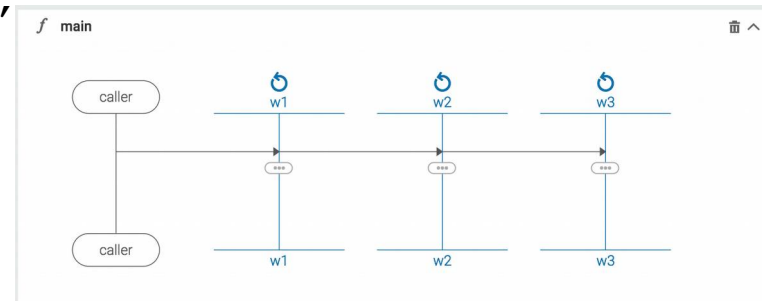
```
import ballerina/io;

function main (string... args) {
    worker w1 {
        io:println("Hello, World! #m");
    }

    worker w2 {
        io:println("Hello, World! #n");
    }

    worker w3 {
        io:println("Hello, World! #k");
    }
}
```

21/06/2018



Ballerina graphical notation

- Sequence diagrams focused on showing worker and endpoint interactions
 - Endpoints represent network services
- Each worker is sequential code
 - We can zoom in graphically but its really just code
- Not meant to be used for low-code / no-code or by non-programmers
 - Meant to make it easier to understand complex distributed interactions

Ballerina, the language

- Program & lexical structure
- Values, types & variables
- Expressions
- Statements
- Workers, Functions, Endpoints & Services
- Tables & streams
- Annotations
- Comments & documentation
- Security
- Distributed resilience

Programs

- Modularity in 4 levels
 - Package repository
 - Organization
 - Package
 - Object, but not mandatory
- Package
 - Collection of files that contribute symbols, like Go
 - File names not relevant
 - Unit of compilation & execution
- Object
 - Usual kind of object

Values, types & variables

- Structurally typed, defined in terms of values
 - Type is a label for a set of values, same value can be of many types
- Kinds of values
 - Simple values: (), boolean, int, float, decimal, string, byte
 - Structured values: tuple, array, map, record, table, xml
 - Behavioral values: function, future, object, stream, typedesc
- Types
 - Basic types (one of the above)
 - Others
 - singleton, union, optional, any, json

Mapping types

- Mappings are collections of (name, value) pairs
- Variations
 - Open vs. closed mappings
 - Required vs. optional fields
- “record” type constructor
 - Collection of mandatory and optional named fields of any types
 - Open if desired
- “map” type constructor
 - Any fields but all values of a single type
 - Always open
- Covariant subtypes
 - Storage type

json

- Widely used format for application level network protocols
- json is just a union
 - () | int | float | string | map<json> | json[]
- json objects are most commonly used
 - map<json>, but subtyped to an open record with non-mandatory fields in most cases
 - Very useful for man-in-the-middle network scenarios

XML

- Widely used format for application level network protocols
- Native data type with literals and easy manipulation

Tables

- Tabular data
 - Programmatically generated & manipulated
 - Database connectors to load/store tables
- Integrated SQL-like query, ala C# LINQ
 - In-memory database
 - Table rows are just records (no ORM)

Streams and streaming query

- Stream is a distributor of any type of events within a BVM
 - Can be attached to network sources as well
- Streaming SQL-like queries attach to some number of streams
 - Complex event processing
 - Event stream processing
 - Marrying Siddhi CEP engine

Type matching

- Union typed expressions have to be separated out before use:

```
match expression {  
  pattern [var] => statement;  
  pattern [var] => statement;  
  ..  
}
```

- Or within an expression:

```
expression but { pattern [var] => expression }
```

Errors and error lifting

- Errors are a built in record type
- Can be returned or thrown
 - Throwing is discouraged and meant to be used rarely
- Error lifting
 - Many functions return `ResultType | error`
 - Check expression: `check expression`
 - If error return immediately, else error is eliminated from type set

Nil & error lifting navigation

- Navigating deep hierarchies is common in integration code
- Navigating through optional types are nil-lifted by “.” operator: a.b.c
- Navigation is both nil and error lifted by “!” operator: a!b!c
- Why?
 - Combination of optional types and type matching makes it impossible to have a null reference in Ballerina
 - Nil & error lifting make those rules palatable

Local & distributed transactions

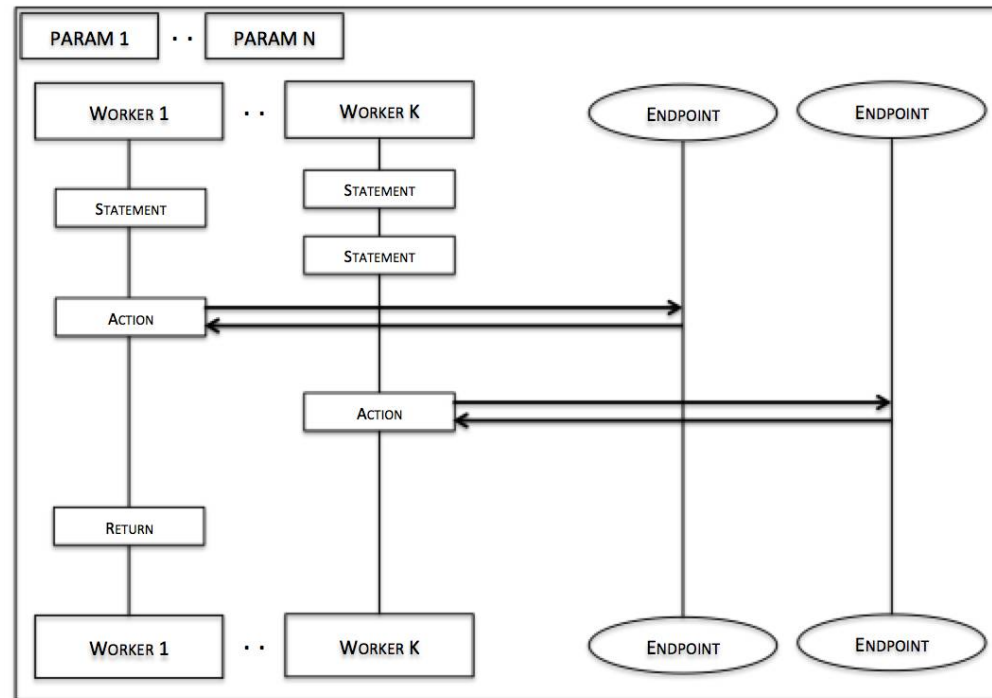
- One service calling another over HTTP is very common in enterprise applications
 - But lack of distributed resiliency leads to fragile code
- Ballerina micro transaction protocol
 - Allows any piece of code to initiate a distributed transaction with its own built-in coordinator
 - Transaction ID is transparently infected to other nodes in the distributed system
 - They join and participate in transaction protocol
- **See:** `transaction { .. } statement`

Writing (more) secure programs

- Objectives
 - Provide integrated authentication and authorization architecture
 - Prevent potentially dangerous data from getting into the system
 - Propagate security contexts to downstream bits
 - Make it easier to monitor access patterns for abuse
- Architecture
 - Runtime context with principal and permissions set by inbound endpoints
 - Pluggable authentication and authorization framework
 - Taint assertions and checking

Functions (and resources)

- Unit of execution: One sequence diagram
- Defines a set of workers who start in parallel when function is invoked
- Any worker can cause caller to be released
 - Other workers continue



Function invocation

- Initialization: start endpoints
- Start concurrent workers
- First one to return will release caller
 - Others continue to completion
- If no one completes (e.g. due to failures) function call fails with “call failed” exception

Futures & non-blocking invocation

- Any function can be called in a non-blocking way:
`future<T> f = start functionName (args)`
- Wait for completion, cancel, get return values via typed future

Worker to worker communication

- Via anonymous channels, non-blocking for send, blocking for receive
- Attempt at deadlock prevention
- Working on named channels

Network calls

- Programmer needs to be made aware that this is a network call
- Syntactic variation:

```
var result = epName->actionName (args);
```
- If any errors occur they need to be type matched and handled

Outbound resiliency

- Failure is normal in network interactions
- Endpoints are logical
 - Group them for load balancing, failover or other behaviors
- Circuit breaking, bulk-heading, timeouts, load management all part of endpoint architecture

Endpoints

- Two kinds: service & client
- Service endpoints are network entry points to a BVM via a registered service
 - Calls are delivered to a particular resource in a service
 - Code can reply/message with provided endpoint, no returning values
- Client endpoints represent remote systems
 - Offer a set of actions for interaction with them

Services & resources

- Service is a collection of resources where each resource is network invocable
- Services must be attached to an endpoint to be invoked
- Responses must be via endpoint and not by returning values
 - Responses may not go

Comments and documentation

- Documentation is a first class aspect of Ballerina
 - Written in markdown+
 - Mandatory for public symbols
- Comments and disabling code

Annotations

- For everything about the code
- Compile-time processable with compiler extensions
 - E.g.: Docker and K8s annotations

BEYOND THE LANGUAGE

Programming is more than just code

- Editing & debugging
- Testing
- Observability
- Dependencies, versions and building
- Documentation
- Sharing

Implementation

- Compiler produces BVM instructions into library (.balo) or linked binary (.balx)
- Extensible architecture for compiler to allow 3rd party annotation processing to become part of compilation process, e.g. Docker/K8s
- IDEs supported via Language Server Protocol
 - Plugins for VS Code, IntelliJ and standalone browser-based Composer
- Compiler & BVM currently written in Java

Future work

- Forward recoverability
- Compensation
- Checkpoint and restart
- Docker / Kubernetes compositions
- Merging / collapsing sequence diagrams
- Internationalizing the grammar
- Implementation
 - Native compilation via LLVM

Summary

- Ballerina is an attempt to build a modern industrial grade programming language
 - For future with lots of network endpoints
- Type system is designed to make network data processing easier
- First class network services along with functions/objects
- Framework to encourage more secure code
- Fully open source and developed openly

Questions?

- Contact: sanjiva@wso2.com
- Download?
 - v0.970.0 released May 1st
 - v0.975.0 about to be released
 - v0.980.0 due in mid-July
 - <http://ballerina.io/>
- Community
 - Email: ballerina-dev@googlegroups.com
 - Slack: [#general](https://ballerina-platform.slack.com)
 - Twitter: @ballerinaplat