



Ballerina

Swan Lake

Modern Compiler Trends: Insights from the Ballerina Compiler

Hasitha Aravinda | hasitha@wso2.com
Architect - WSO2

Why Are You Studying Compiler Theory?





Why Might You Build or Work on a Compiler in the Future?

What Defines a Compiler Today?

A glowing lightbulb is centered in the background, emitting a soft light that illuminates the surrounding teal gradient. The lightbulb's filament is visible and appears to be lit, creating a bright point of light within the bulb. The overall aesthetic is clean and modern, with a focus on the lightbulb as a symbol of ideas or innovation.

Modern Compilers Are No Longer Standalone Applications

Compiler Ecosystem

- IDEs. i.e. VSCode, IntelliJ Idea
- Code Scanners
 - Security Scanners:
 - Linters, Spot Bug, Sonar Scan, Check Style
- Tools
 - Code Formatters, Code Visualizers, Artifact Generators, Report Generators
- Package Management System
- Cross-Compilers

and many more...

Modern Compiler Trends

LSP

Language Server Protocols

LSP

- Created by Microsoft
- Standardize the protocol for how language development tools communicate with each other.
- Encourage reusability.
- Features
 - Auto complete
 - Go to definition
 - Documentation on hover
 - Refactoring

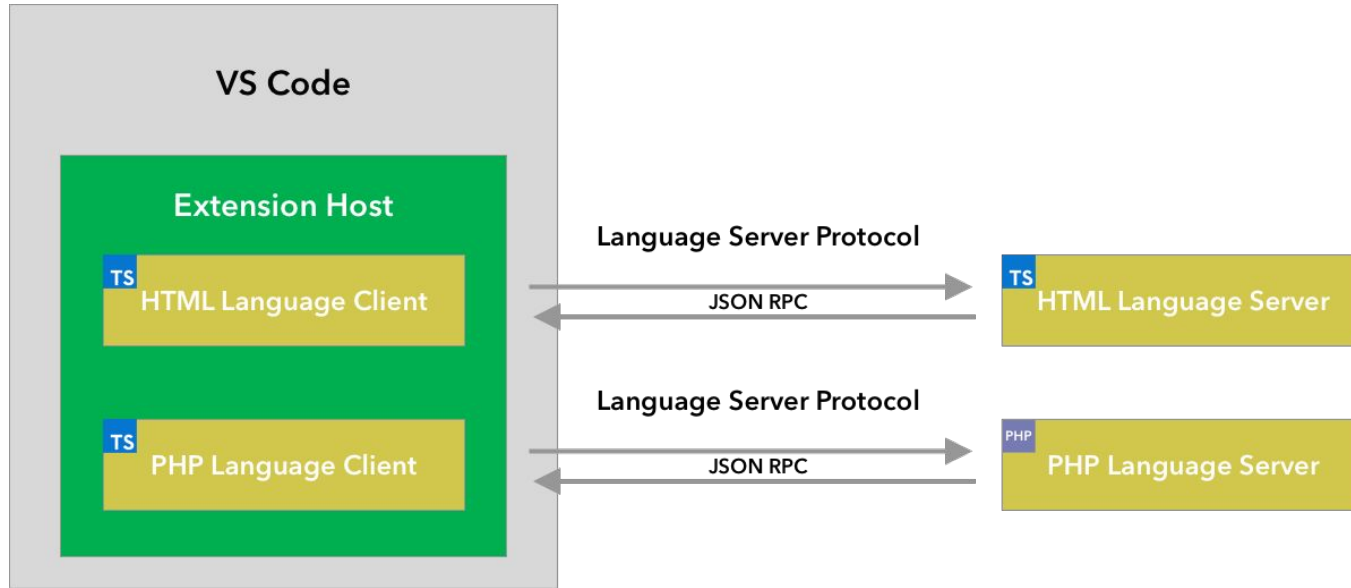
and many more...

LSP



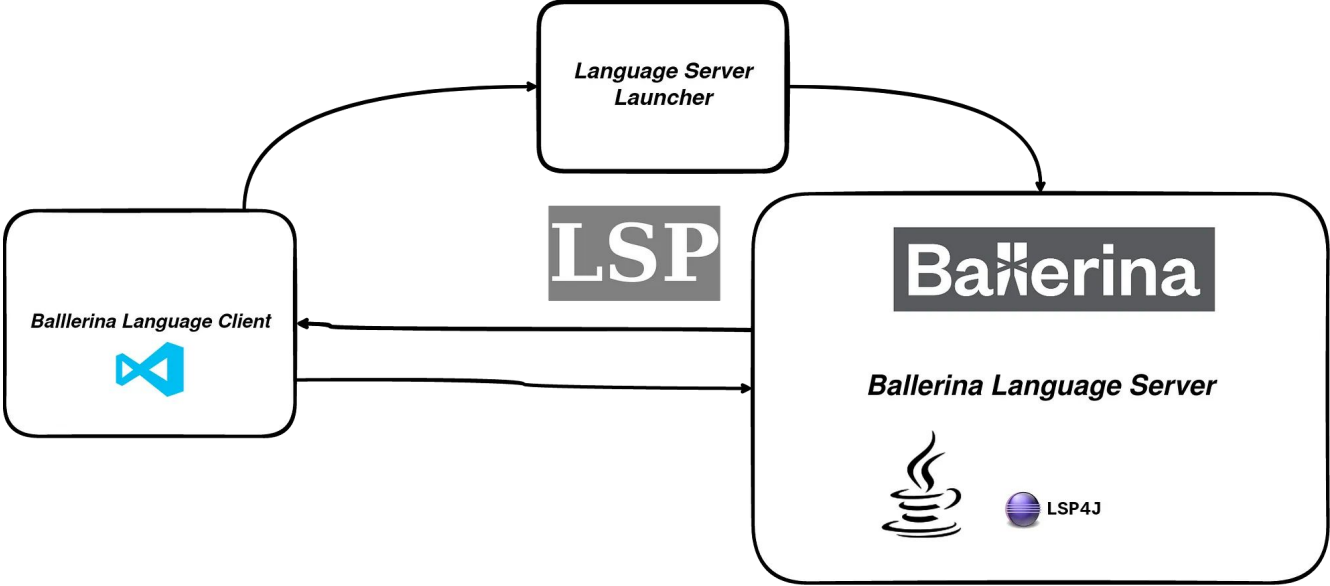
Credits: <https://code.visualstudio.com/api/language-extensions/language-server-extension-guide>

LSP



Credits: <https://code.visualstudio.com/api/language-extensions/language-server-extension-guide>

Ballerina LSP



Credits: <https://medium.com/ballerina-techblog/practical-guide-for-the-language-server-protocol-3091a122b750>

WebAssembly (Wasm)

Wasm

- **Binary Instruction Format:** Wasm is a binary instruction format designed for a stack-based virtual machine.
- **Compilation Target:** It serves as a portable target for compiling high-level languages like C, C++, and Rust, enabling them to run on the web.
- **Performance:** Provides near-native execution speed, making it suitable for performance-critical applications such as games and complex UIs.
- **Security:** Designed to maintain the security guarantees of the web, ensuring safe execution within a sandboxed environment in web browsers.
- **Platform Independence:** Wasm is platform-independent, facilitating consistent behavior across different systems and devices.

Examples

- Run Linux or other Operating Systems in your browser!
 - <https://bellard.org/jslinux/>
- Web AutoCad
 - <https://web.autocad.com/>
- RustPython
 - <https://rustpython.github.io/demo/>
- More Applications: <https://madewithwebassembly.com/>

AI and Machine Learning

AI and Machine Learning

- Multiple Areas
 - Context aware completion
 - Bug detection
 - Code optimization
 - Natural language to Code
 - Personalization

Tools

- VSCode Copilot
- <https://cursor.sh/>
- OpenAI's ChatGPT and APIs

Many more

Cloud Based Development

Cloud Based Development

- Developer platform as a service
- Cloud compilers as a service

- Advantages
 - Accessibility, Collaboration, Scalability, Reduced Hardware Cost
- Disadvantages
 - Internet dependency, Security, Limited features, Performance, Cost over time.
- Products
 - Github Actions
 - Github Codespaces
 - AWS Cloud9
 - WSO2 Choreo

Other Trends

- Cross-Language Interoperability
- DSL
- Quantum Computing Language Development
- Adaptive Optimization
- Energy Efficient
- Secure Compilation

and more ...

Extending Ballerina Compiler

Compilation Phases As Discussed In The Dragon Book

Error Handling and Reporting

Lexical
Analysis



Syntax
Analysis



Semantic
Analysis



Intermediate
Code Gen



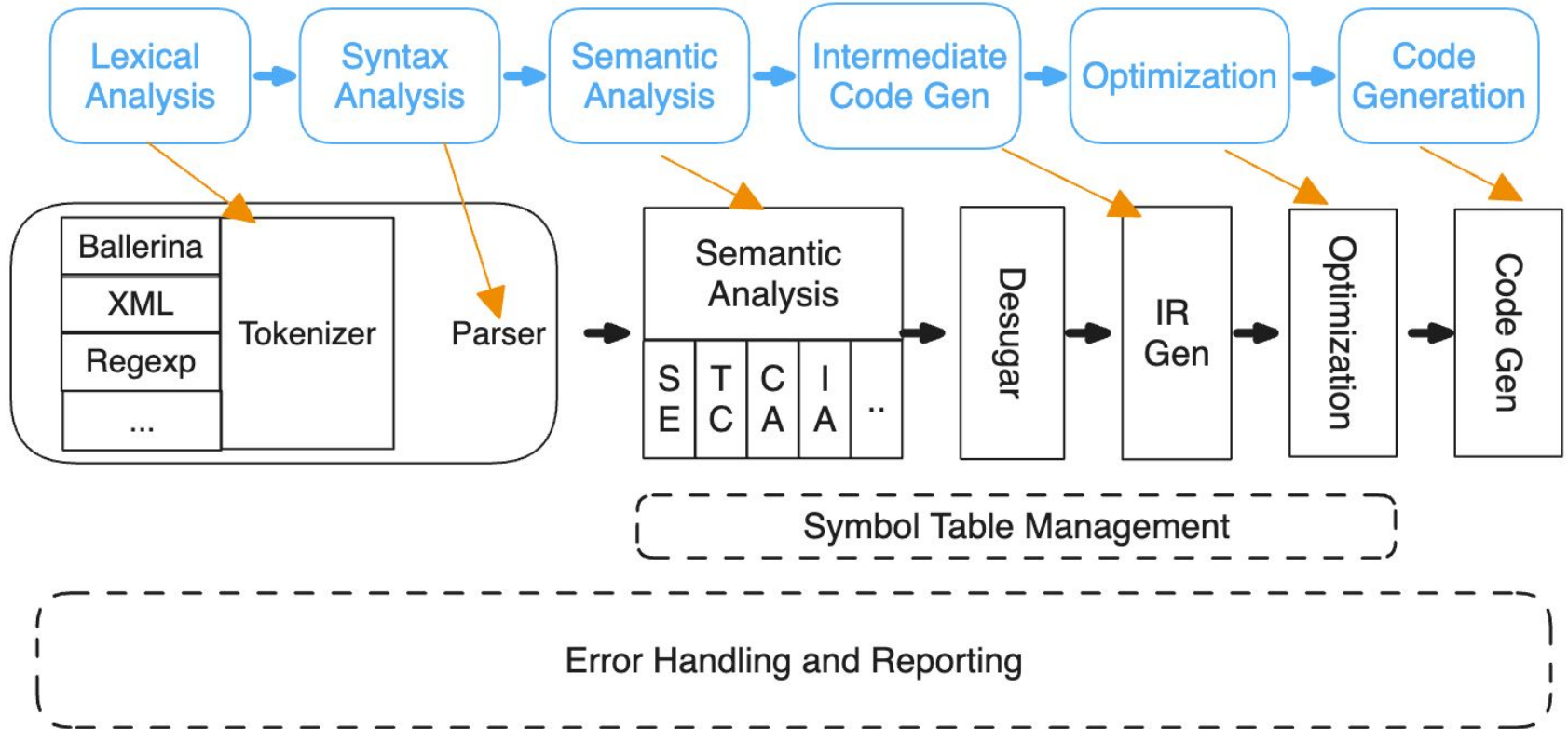
Optimization



Code
Generation

Symbol Table Management

Ballerina Compiler - Compilation Phases



Feature Aware Mini Phases

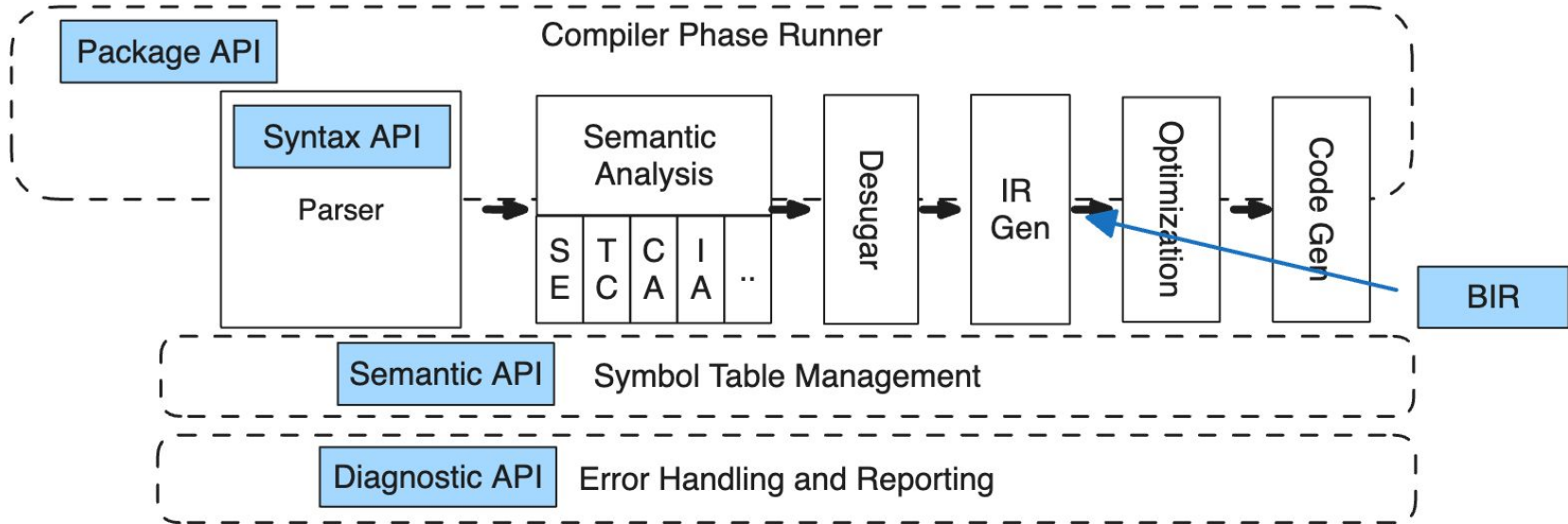
- Rather than having single monolithic phase, we break phases into small phases based on requirements
 - Semantic Analysis
 - Constant Analysis - resolving compile time constants
 - Symbol Enter - resolving module level constructs, types, variables, functions, etc.
 - Worker Analysis - resolving worker(concurrent constructs) interactions
 - Semantic Analysis* - resolving statements
 - Type Checker - resolving expressions
 - Code Analysis - resolving reachability
 - Isolate Analysis - checking concurrency.
- and more.

Interutable Phases

- Redesign Phases such that, we can tell upto which point compilation should run.
- Introduced a compilation phase runner to manage the compilation requests.

Reusable Phases and Results

- Redesign phases such that, result of each phase can be reuse.
- Introduced Compiler APIs.
- Use Generated intermediate code (**BIR**) as the lowered version.

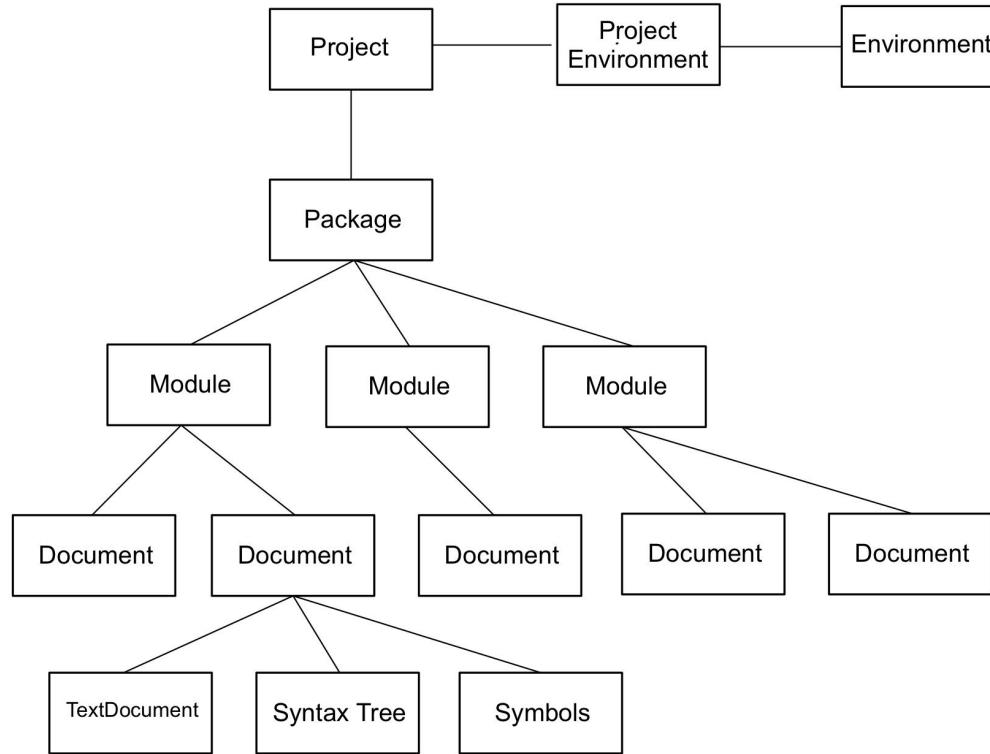


Use Cases

Use Case 1: Supporting Multiple Modules

- Package is a logical source code grouping.
- Developers compile a project (using CLI).
- A package contains one or more ballerina modules.
- Modules are reusable components. i.e `import ballerina/http`
- A module is Ballerina's unit of compilation (CompilationUnit).
 - Compiler phases run against a module.
- Package Structure
 - Parent module
 - Zero or More sub Modules
- Modules can have inter-dependencies, but no cyclic.
- One executable per package, that is for the parent module.

Use Case 1: Supporting Multiple Modules



Use Case 1: Supporting Multiple Modules

Example: ballerina/graphql [package](#)

- graphql
- graphql.dataloader
- graphql.subgraph

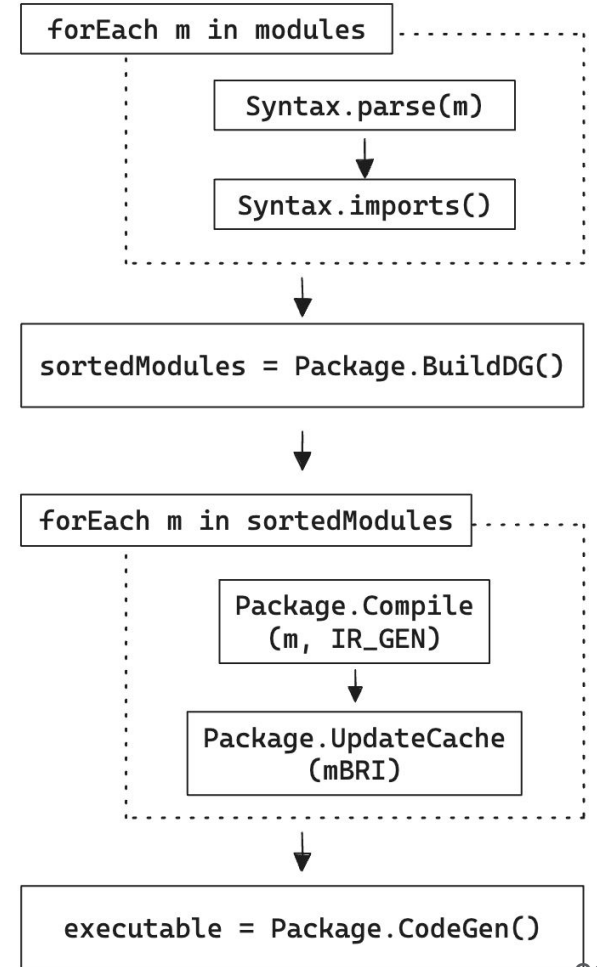
Problems we have to solve.

- Multiple compilation units \Rightarrow Multiple compilations
- Decide compilation order
- Reusable Symbol Table Entries
- Single executable \Rightarrow Merge compilation results.

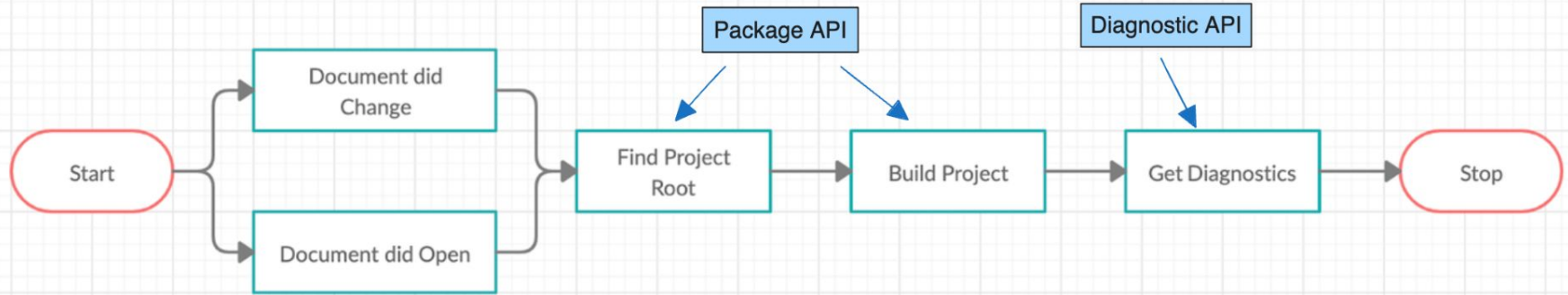
Use Case 1: Supporting Multiple Modules

Our Solution

- Use the syntax API to get list of import statements for each module.
- Use this data build the dependency graph
- Execute the compilation phases upto BIR generation, from bottom to top of the dependency graph.
- Reuse BIRs of compiled modules
- At the parent module, execute full compilation to get the executable.

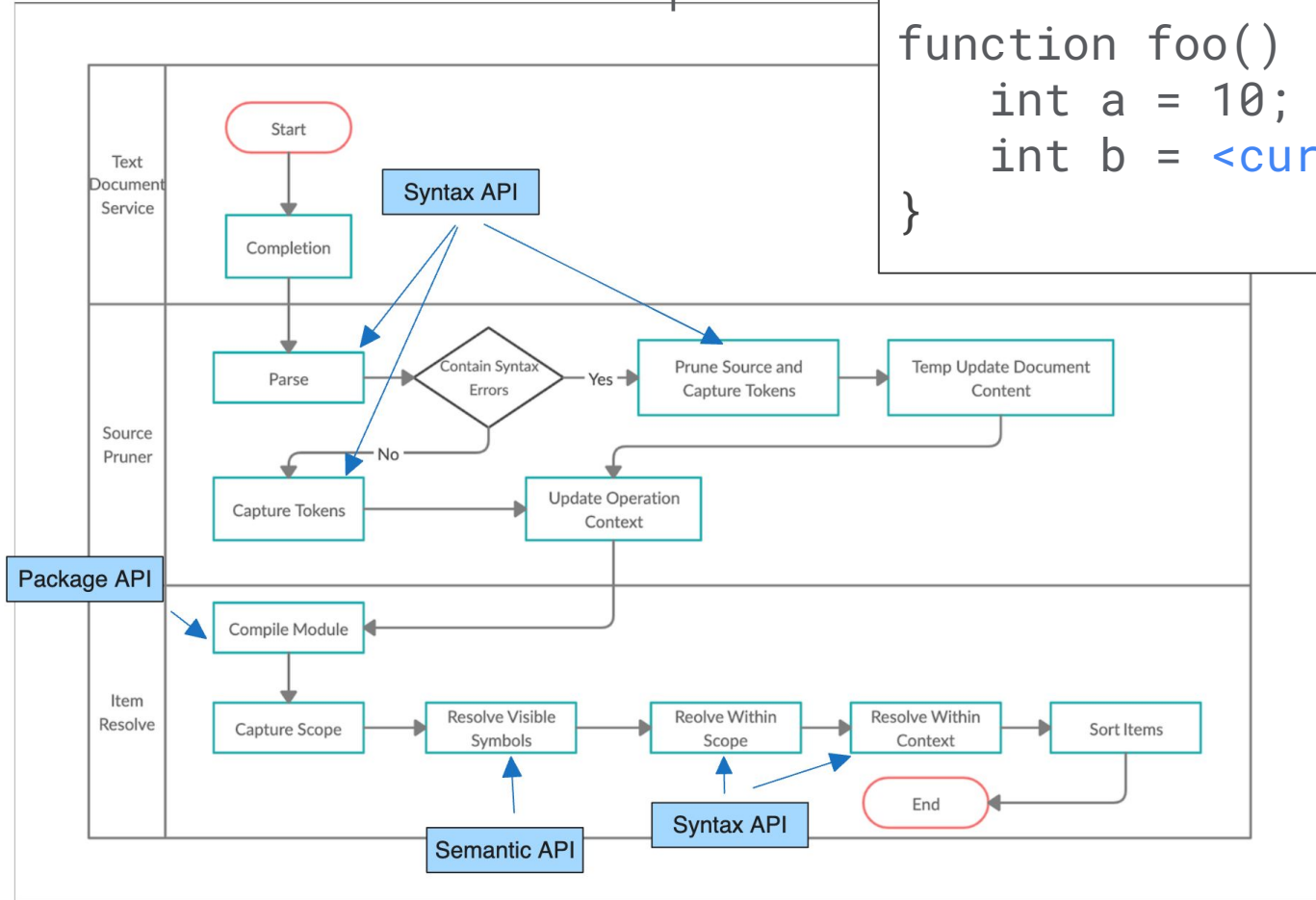


Use Case 2: VSCode Show Diagnostics



Use Case 3: VSCode Code Completion

```
function foo() {  
  int a = 10;  
  int b = <cursor>  
}
```

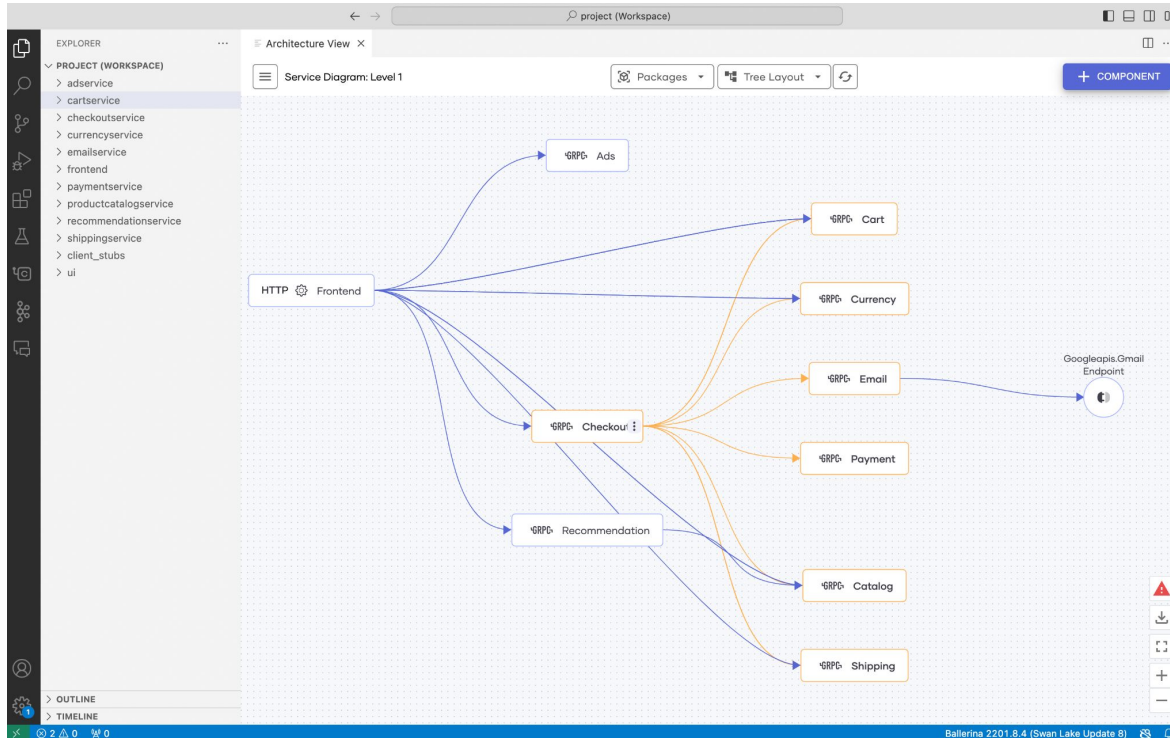


Use Case 4: Compiler Extensions

- Extend language semantics
- Kinds
 - Validate Code
 - Modify Code
 - Generate New Code and Artifacts
- Examples
 - Code Validation
 - Http resource function can have only certain parameters.
 - Validate annotation attachments.
 - Code Modifications
 - Attached OpenAPI specification as attachment.
 - Code Generation
 - Generate docker and k8s artifacts.

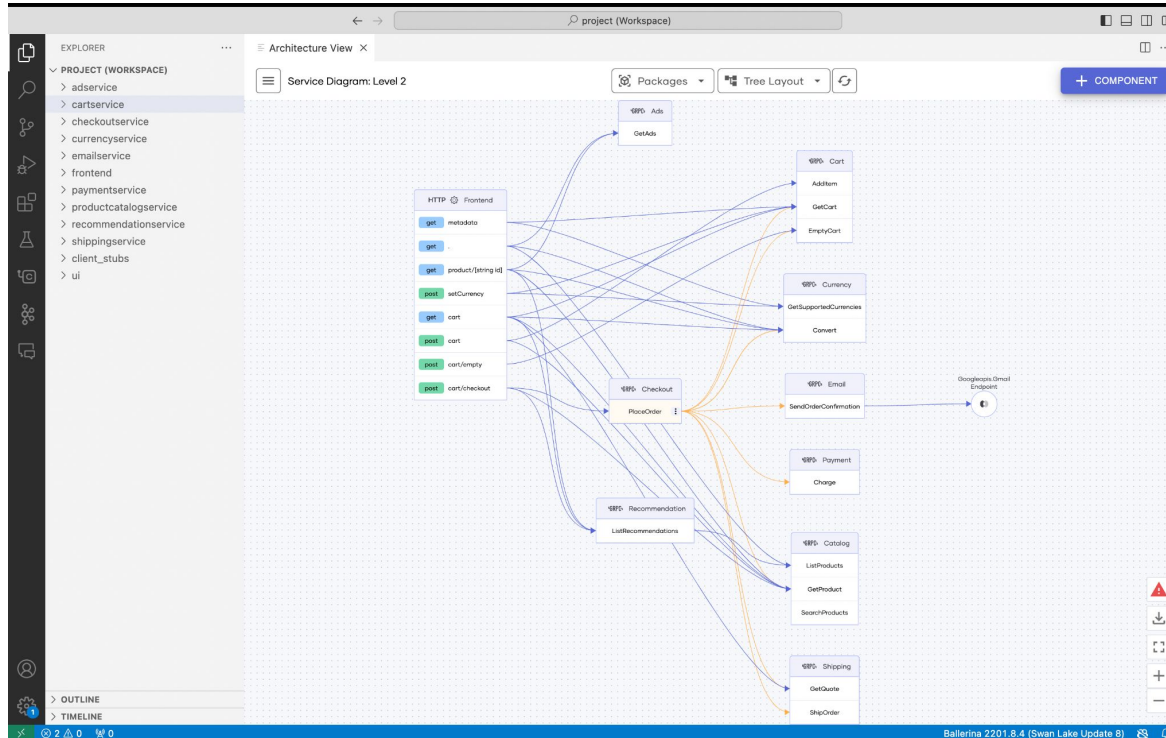
Use Case 5: Visual Features

Architectural design view - Services



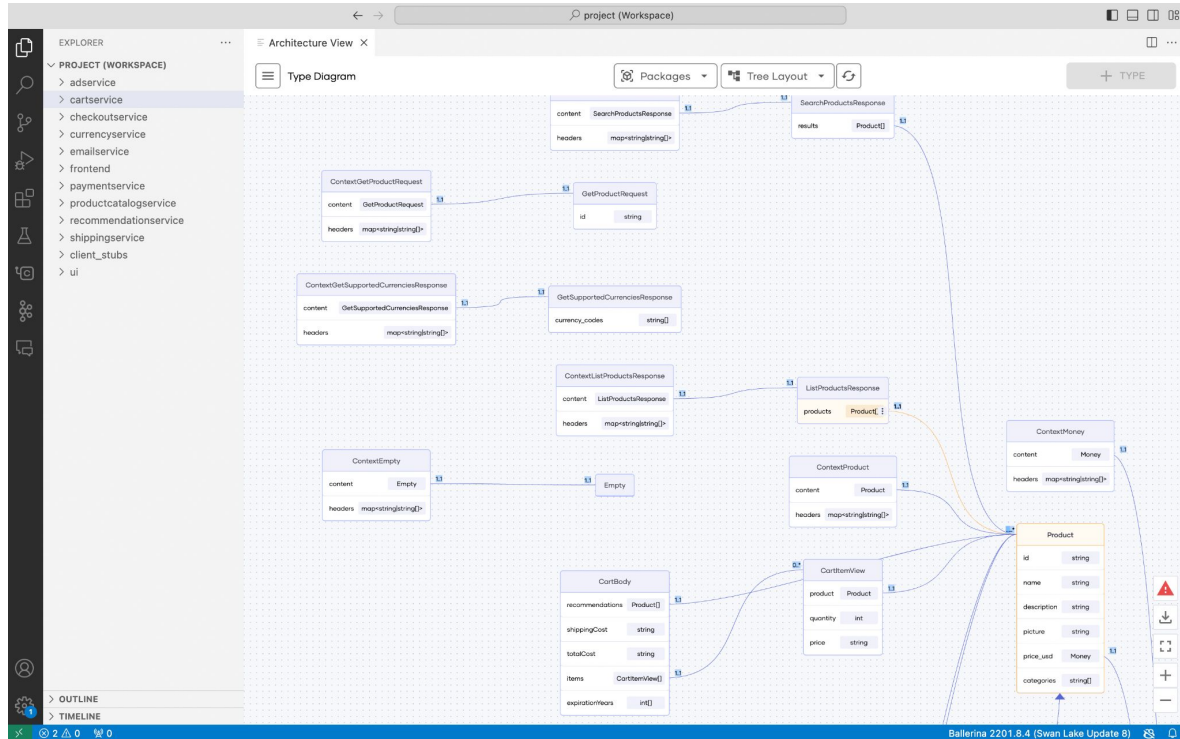
Use Case 5: Visual Features

Architectural design view - Resources



Use Case 5: Visual Features

Architectural design view - Types



Use Case 5: Visual Features

Service designing

The image displays a development environment for designing REST APIs. It is divided into three main sections:

- Code Editor (Left):** Shows the source code for a REST API service in `service.bal`. The code defines a `table` of albums and three REST API endpoints: `get albums`, `get albums/{string id}`, and `delete path/{string id}`.
- Overview Diagram (Middle):** Provides a visual summary of the service components. It lists:
 - Services:** A root service `/`.
 - Types:** A type `(i) Album`.
 - ModuleVariables:** Variables `port` and `albums`.
- Configure Resource (Right):** A dialog for configuring a specific resource. It shows:
 - Service:** `/ listening on new httpListener{port}`
 - HTTP Method:** `GET`
 - Resource Path:** `path`
 - Parameters:** A list of methods: `GET albums`, `GET albums[{string id}]`, `POST albums`, and `DELETE path/{string id}`.
 - Responses:** A response `500 error?`.

Use Case 5: Visual Features

Data mapping

The screenshot displays a development environment with two main panels. The left panel shows a code editor with Go code for a service named `calendar`. The code includes functions for handling new events, deletions, and updates, with comments in Italian. A design section defines a `calEventToTrelloCard` function that maps fields from a `calendar.Event` to a `TrelloCard`.

```
52 Run | Debug | Try It
53 service calendar:CalendarService on calendarListener {
54     remote function onNewEvent(calendar:Event payload) returns error {
55         do {
56             // Add the card to the Trello list
57             trello:Cards card = calEventToTrelloCard(payload);
58             _ = check trello->addCards(card);
59
60             // Send SMS notification
61             string twilioMsg = calEventToMessage(payload);
62             _ = check twilio->sendSms(twFromMobile, twToMobile, twMsg);
63
64         } on fail var e {
65             // Log the error and add the event to the dead letter channel
66             log:printError(string "Failed to process the calendar event: ", e);
67             toDeadLetterChannel(payload, e);
68         }
69     }
70
71     remote function onEventDelete(calendar:Event payload) returns error {
72     }
73
74     remote function onEventUpdate(calendar:Event payload) returns error {
75     }
76 }
77
78 Design
79 function calEventToTrelloCard(calendar:Event calEvent) returns trello:Cards {
80     name: calEvent.summary,
81     due: calEvent.end?.dateTime,
82     idList: trelloListId,
83     desc: string "New event is created on Google Calendar: ${calEvent.summary}
84         The event starts on ${calEvent.start?.dateTime ? ""} and ends on
85         ${calEvent.end?.dateTime ? ""}";
86
87     Design
88     function calEventToMessage(calendar:Event calEvent) returns string {
89         string "New event is created : ${calEvent.summary ? ""} starts
90         ends on ${calEvent.end?.dateTime ? ""}";
91     }
92 }
93
94 > function toDeadLetterChannel(calendar:Event calEvent, error e) {
95     log:printError(string "Failed to process the calendar event: ", e);
96     toDeadLetterChannel(payload, e);
97 }
```

The right panel shows the 'Data Mapper' tool for the `calEventToTrelloCard` function. It visualizes the mapping between the source `calEvent: trigger.google.calendar.Event` and the target `trello:Cards`. The source fields include `kind`, `etag`, `id`, `status`, `htmlLink`, `created?`, `updated?`, `summary?`, `description?`, `location?`, `calendar?`, `creator?`, `organizer?`, `start?`, `date?`, `dateTime?`, `timeZone?`, `end?`, `date?`, `dateTime?`, `timeZone?`, `endTimeUnspecified?`, `recurrence?`, `recurrenceStart?`, and `recurrenceEnd?`. The target fields include `closed`, `desc`, `due`, `fileSource`, `idAttachmentCover`, `idBoard`, `idCardSource`, `idLabels`, `idList`, `idMembers`, `keepFromSource`, `labels`, `name`, `pos`, `subscribed`, and `urlSource`. Blue lines connect the source fields to their corresponding target fields, illustrating the data mapping process.

Use Case 5: Visual Features

Data persistence

```
1 import ballerina/time;
2 import ballerina/persist as _;
3
4 type User record {
5     readonly string id;
6     string username;
7     string fullname;
8     string role;
9     Challenge[] challenge;
10    Contest[] moderatedContests;
11    Submission[] submissions;
12 };
13
14 type Contest record {
15     readonly string id;
16     string title;
17     byte[] readmeFile;
18     time:Civil startTime;
19     time:Civil endTime;
20     string imageUrl;
21     ChallengesOnContests[] challenges;
22     Submission[] submissions;
23     User moderator;
24 };
25
26 type Challenge record {
27     readonly string id;
28     string title;
29     time:Civil createTime;
30     byte[] templateFile;
31     byte[] readmeFile;
32     string difficulty;
33     byte[] testCasesFile;
34     ChallengesOnContests[] contests;
35     User author;
36     Submission[] submissions;
37 };
```

The Entity Relationship Diagram (ERD) visualizes the following entities and their relationships:

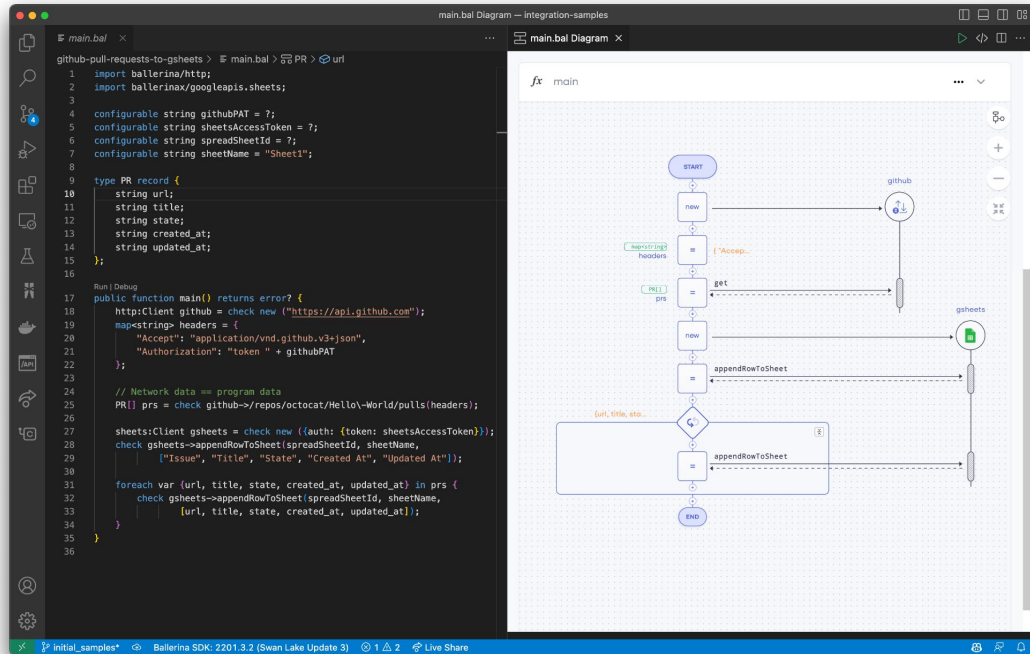
- Challenge**: Attributes include `id` (string), `title` (string), `createTime` (Civil), `templateFile` (byte[]), `readmeFile` (byte[]), `difficulty` (string), `testCasesFile` (byte[]), `contests` (ChallengesOnContests[]), `author` (User), and `submissions` (Submission[]).
- Contest**: Attributes include `id` (string), `title` (string), `readmeFile` (byte[]), `startTime` (Civil), `endTime` (Civil), `imageUrl` (string), `challenges` (ChallengesOnContests[]), `submissions` (Submission[]), and `moderator` (User).
- ChallengesOnContests**: Attributes include `id` (string), `challenge` (Challenge), `contest` (Contest), and `assignedTime` (Civil).
- User**: Attributes include `id` (string), `username` (string), `fullname` (string), `role` (string), `challenge` (Challenge[]), `moderatedContests` (Contest[]), and `submissions` (Submission[]).
- Submission**: Attributes include `id` (string), `submittedTime` (Civil), `score` (float), `challenge` (Challenge), `contest` (Contest), and `user` (User).

Relationships are shown with cardinalities:

- Challenge** (1) to **ChallengesOnContests** (0..*): 1:0..*
- Challenge** (1) to **Contest** (1): 1:1
- Challenge** (1) to **Submission** (0..*): 1:0..*
- Contest** (1) to **ChallengesOnContests** (0..*): 1:0..*
- Contest** (1) to **Submission** (0..*): 1:0..*
- Contest** (1) to **User** (1): 1:1
- User** (1) to **Submission** (0..*): 1:0..*
- User** (1) to **Challenge** (1): 1:1

Use Case 5: Visual Features

Text and graphical syntax parity



Key Lessons from Developing Ballerina Lang

- It is a Platform
- Continuous Adaptation
- Incremental vs. Radical Change
- User-Centric Development

Find out more...

- Learn Ballerina
 - [Learn pages](#)
 - [Ballerina by example](#)
 - [Ballerina VS Code extension](#)
 - [Ballerina training video series](#)
 - [Ballerina certification](#)
- Join the Ballerina community



[ballerinalang](#)



[Tag : ballerina](#)



[@ballerinalang](#)



[ballerina-lang](#)

Questions?

