

Ballerina

Swan Lake

Modern Compiler Trends: Insights from the Ballerina Compiler

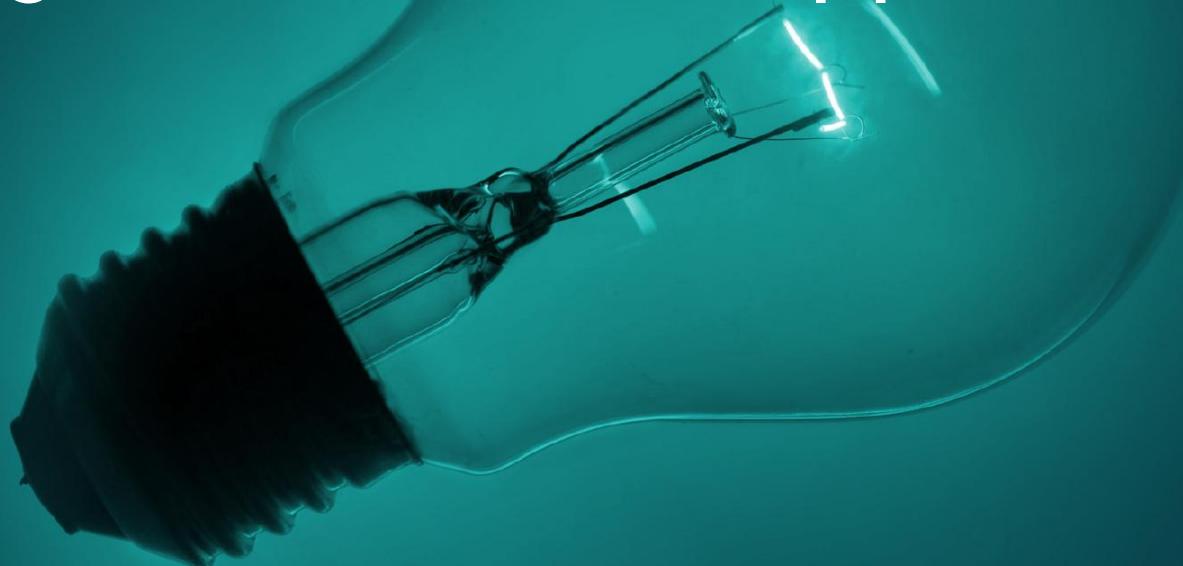
Hasitha Aravinda | hasitha@wso2.com
Architect - WSO2

Why Are You Studying Compiler Theory?

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

What Defines a Compiler Today?

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

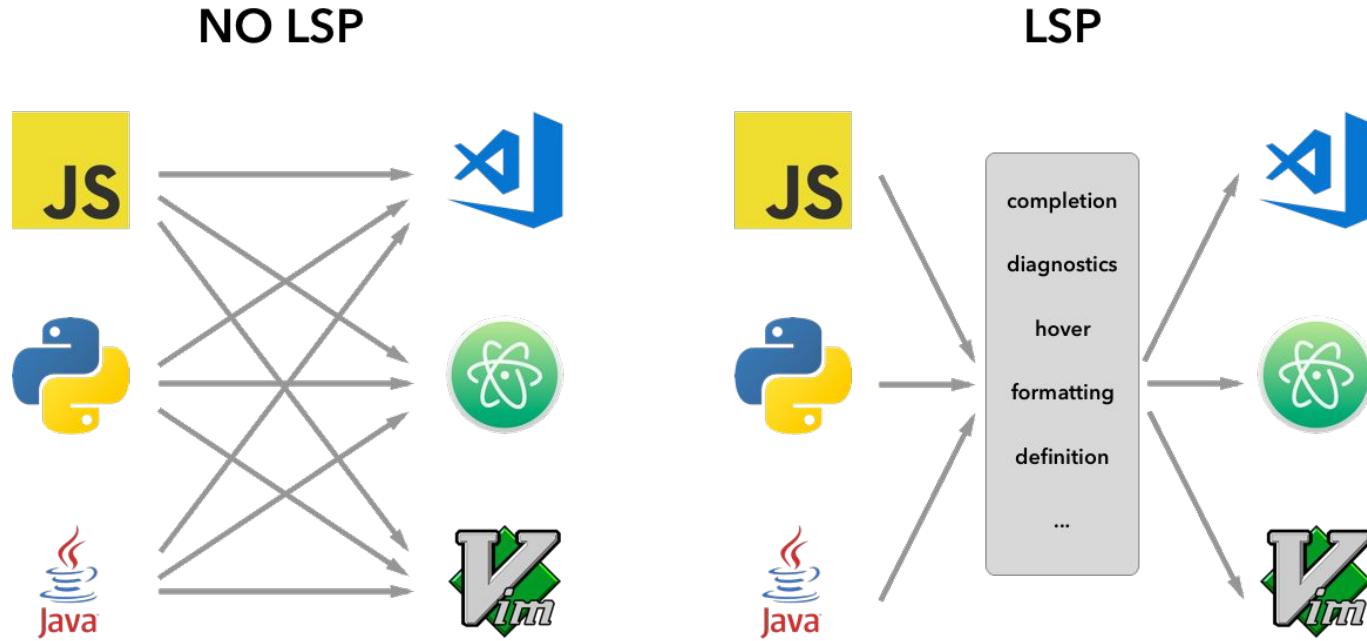
LSP

Language Server Protocols

LSP

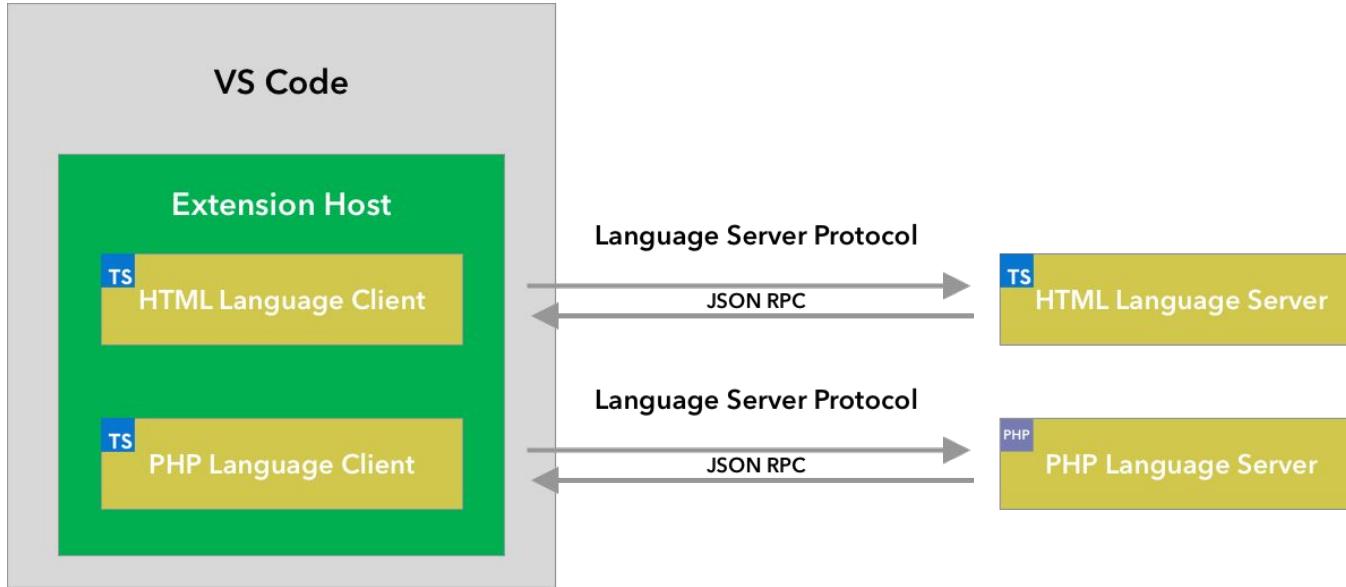
- Created by Microsoft
 - Standardize the protocol for how language development tools communicate with each other.
 - Encourage reusability.
 - Features
 - Auto complete
 - Go to definition
 - Documentation on hover
 - Refactoring
- and many more...

LSP



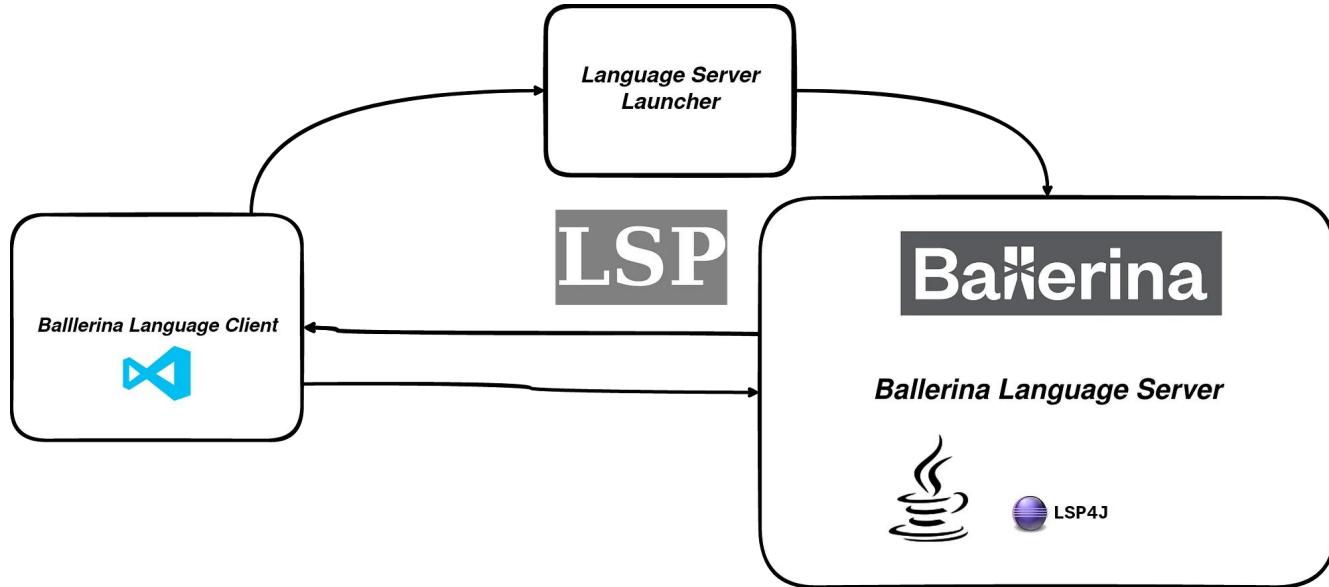
Credits: <https://code.visualstudio.com/api/language-extensions/language-server-extension-guide>

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

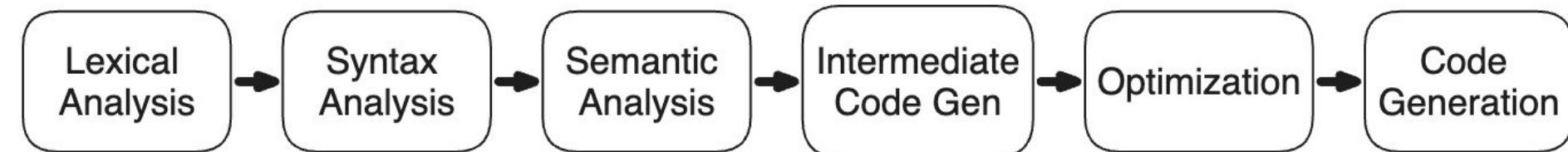
- Cross-Language Interoperability
- DSL
- Quantum Computing Language Development
- Adaptive Optimization
- Energy Efficient
- Secure Compilation

and more ...

Extending Ballerina Compiler

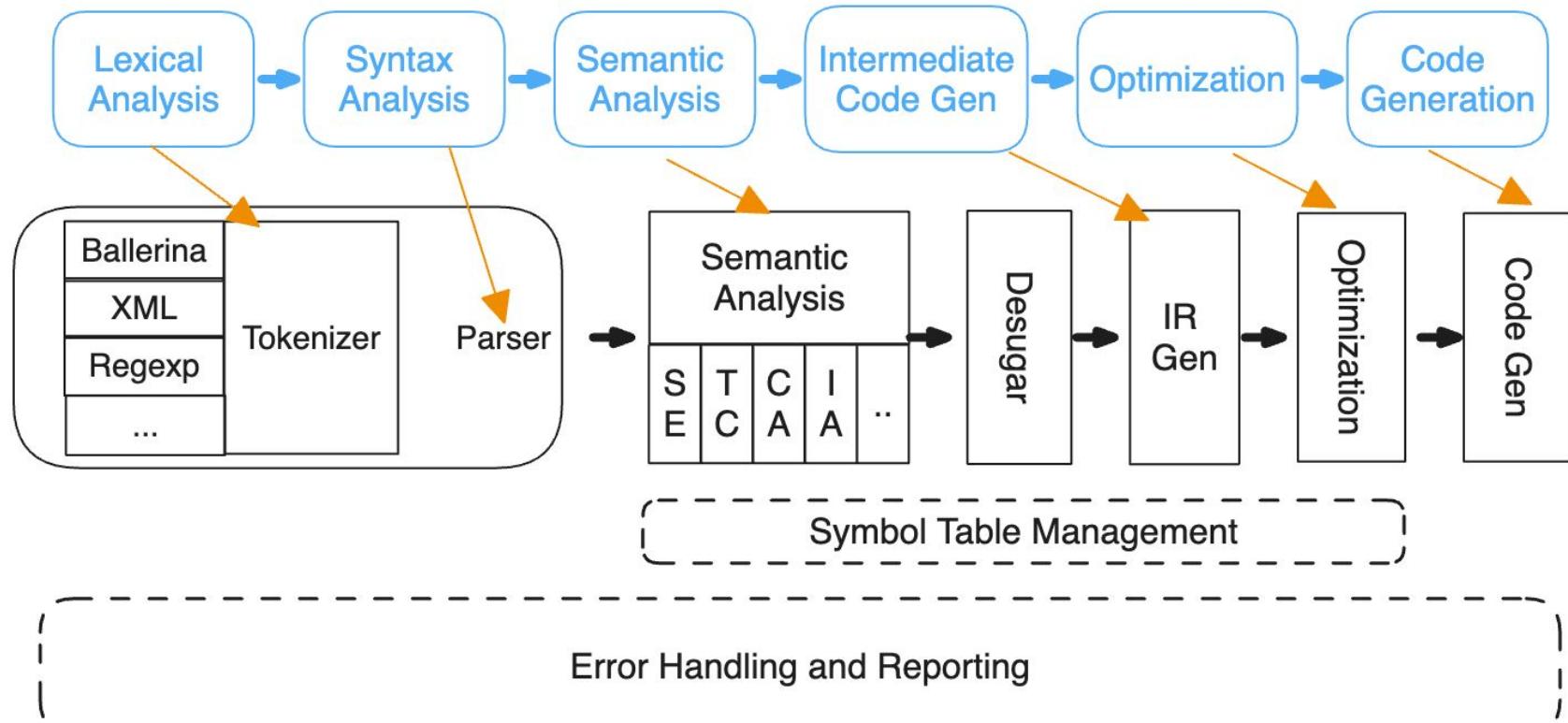
Compilation Phases As Discussed In The Dragon Book

Error Handling and Reporting



Symbol Table Management

Ballerina Compiler - Compilation Phases



Feature Aware Mini Phases

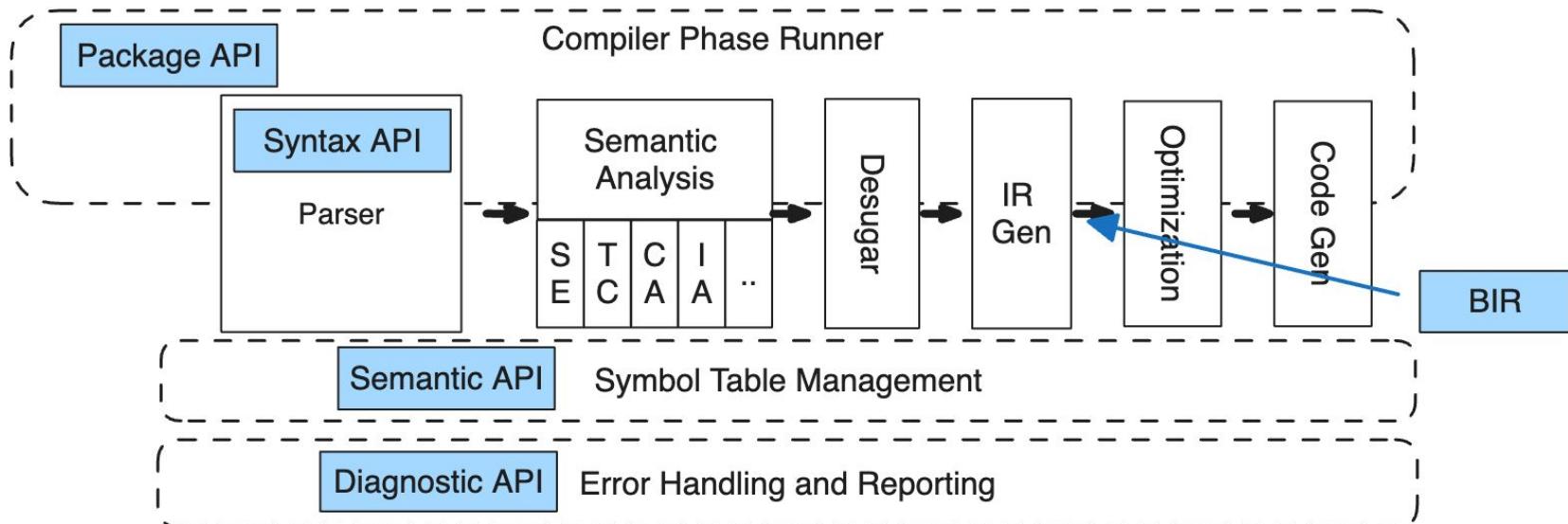
- Rather than having single monolithic phase, we break phases into small phases based on requirements
- Semantic Analysis
 - Constant Analysis - resolving compile time constants
 - Symbol Enter - resolving module level constructs, types, variables, functions, etc.
 - Worker Analysis - resolving worker(concurrent 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.

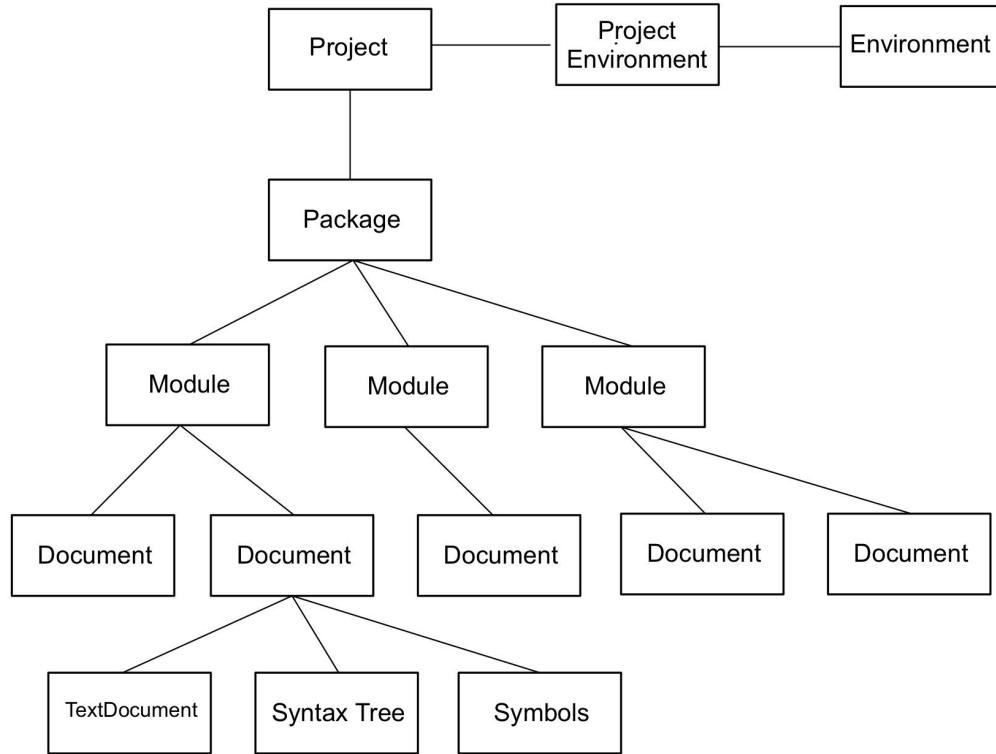


Use Cases

Use Case 1: Supporting Multiple Modules

- Package is a logical source code grouping.
- Developers compiles 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

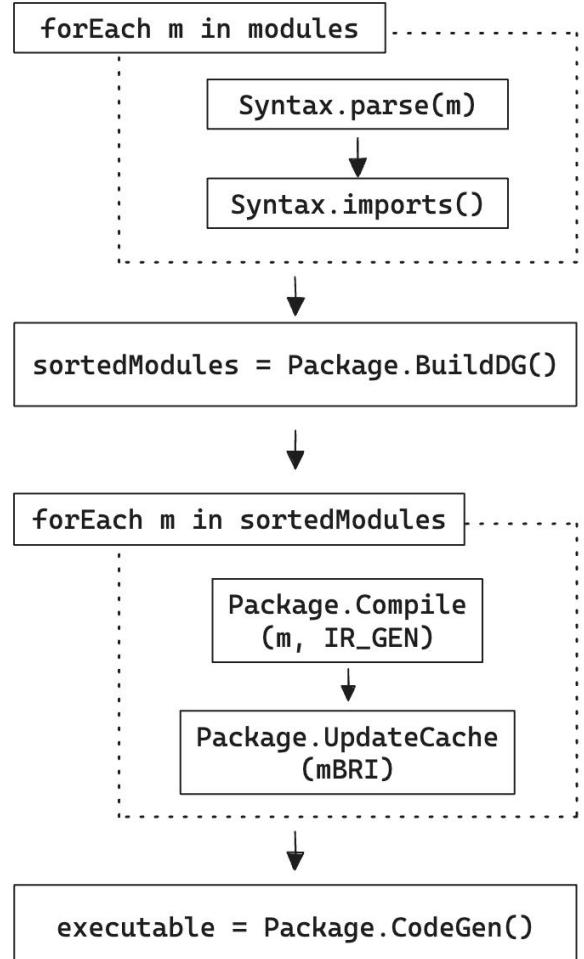
Problems we have to solve.

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

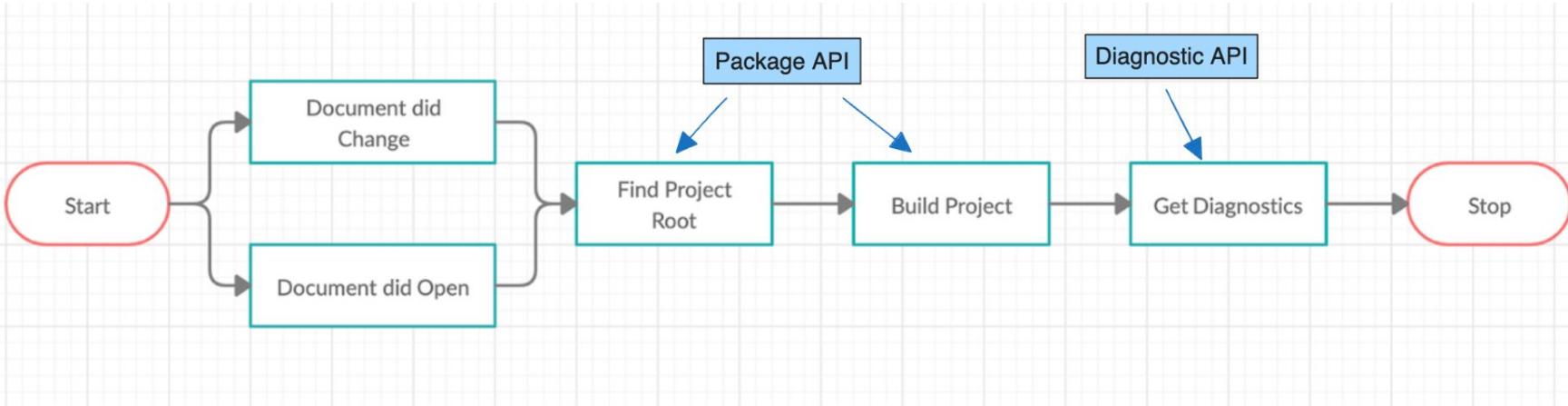
Use Case 1: Supporting Multiple Modules

Our Solution

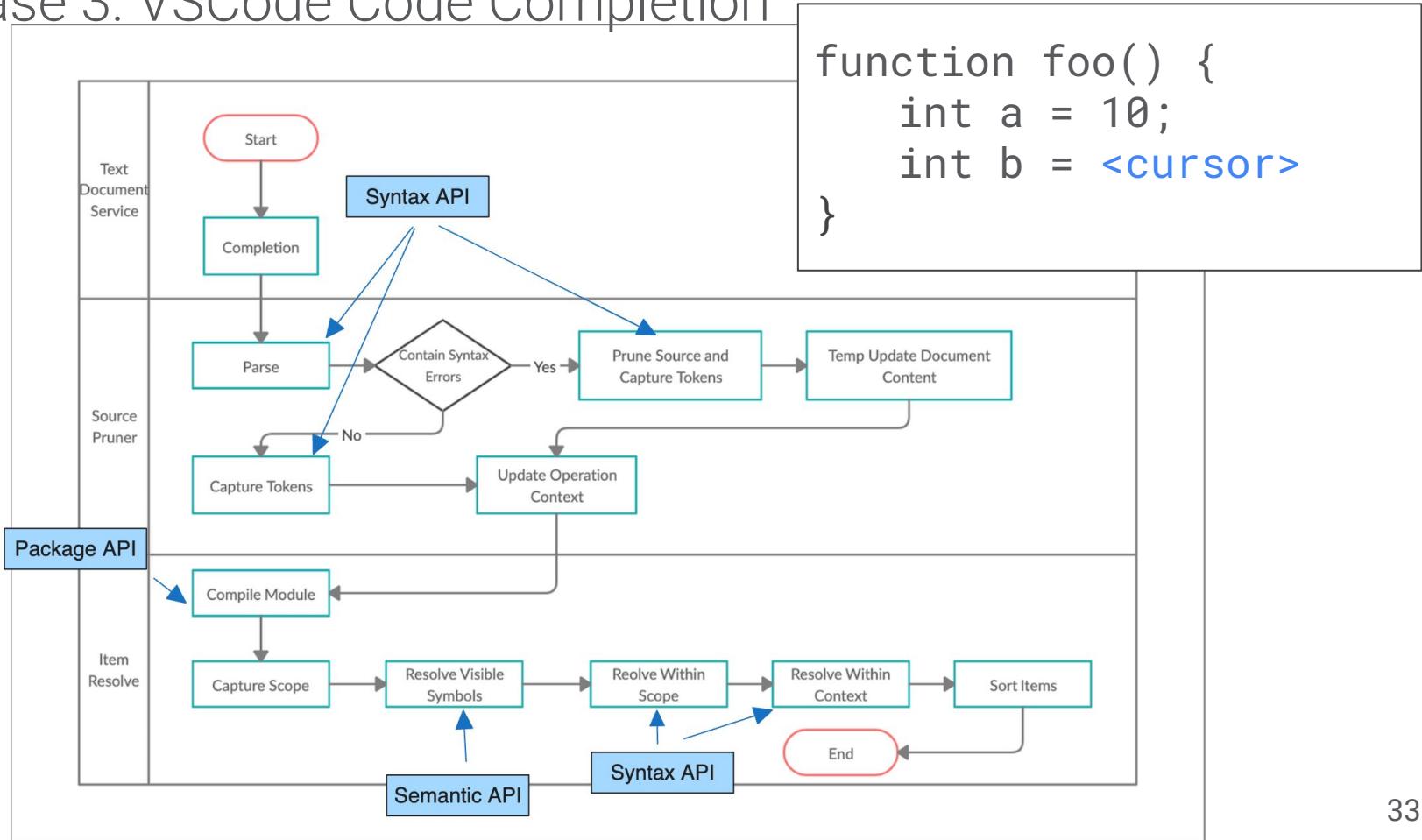
- Use the syntax API to get list of import statements for each module.
- Use this data build the dependency graph
- Execute the compilation phases upto BIR generation, from bottom to top of the dependency graph.
- Reuse BIRs of compiled modules
- At the parent module, execute full compilation to get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



Use Case 2: VSCode Show Diagnostics



Use Case 3: VSCode Code Completion

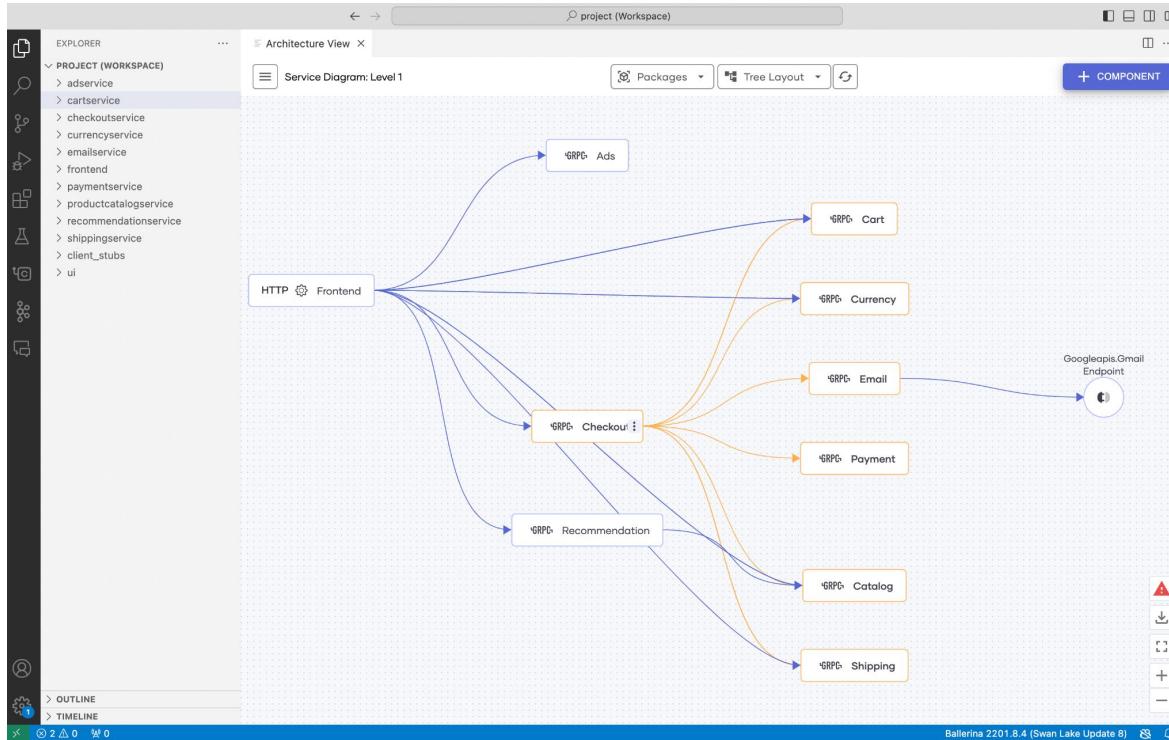


Use Case 4: Compiler Extensions

- 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.

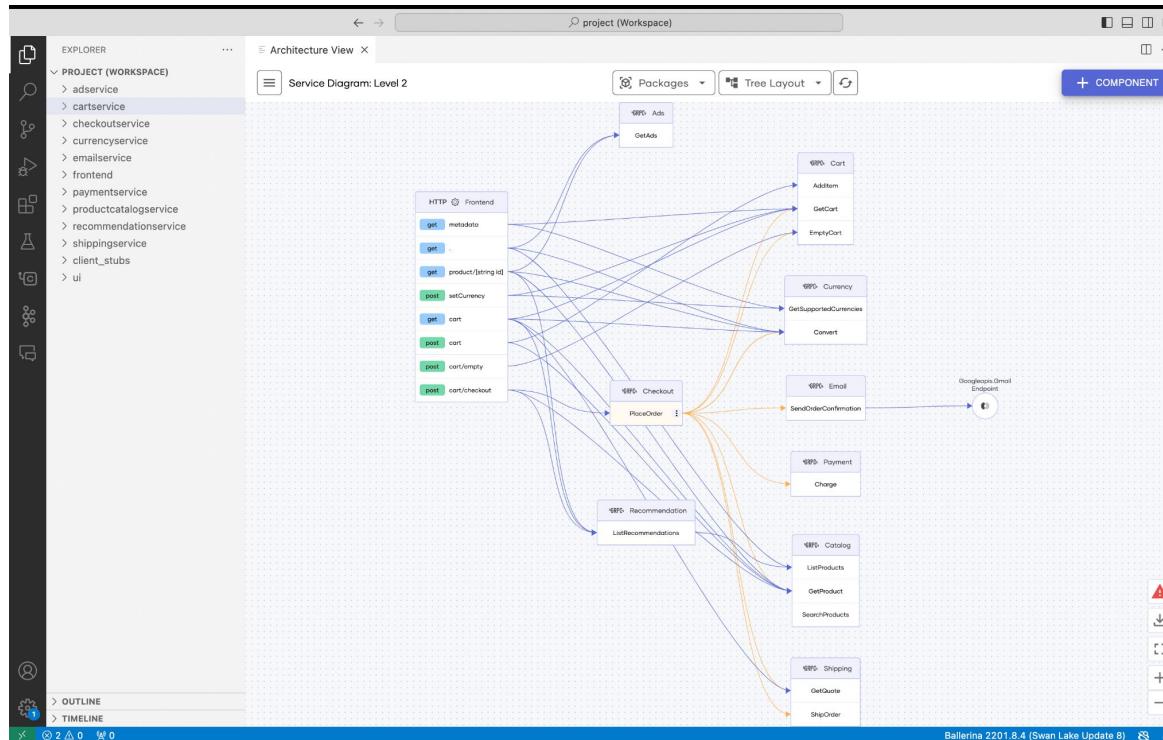
Use Case 5: Visual Features

Architectural design view - Services



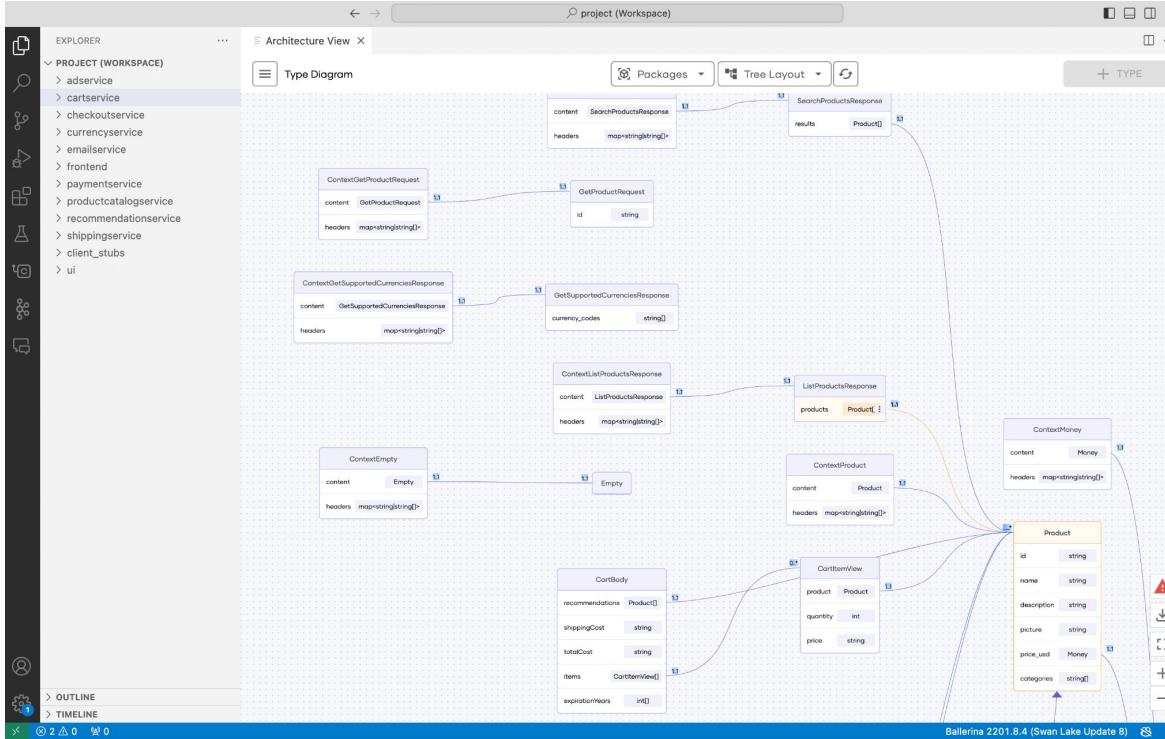
Use Case 5: Visual Features

Architectural design view - Resources



Use Case 5: Visual Features

Architectural design view - Types



Use Case 5: Visual Features

Service designing

The screenshot displays the Ballerina IDE interface with several windows open, illustrating the visual features for service designing:

- Left Sidebar (Explorer):** Shows the project structure with files like `service.bal`, `target`, and configuration files.
- Code Editor (service.bal):** Displays the Ballerina code for a service. It includes:
 - A `RESTAPI` target definition.
 - A `service.bal` file containing:
 - Imports: `import ballerina/http;`, `import ballerina/ignore;`
 - Configurable port declaration: `configurable int port = 8080;`
 - Type definition: `type Album readonly & record { string id; string title; string artist; decimal price; };`
 - Table definition: `table<Album> key(id) albums = table [{id: "1", title: "Blue Train", artist: "John Coltrane", price: 56.99}, {id: "2", title: "Jero", artist: "Gerry Mulligan", price: 17.99}, {id: "3", title: "Sarah Vaughan and Clifford Brown", artist: "Sarah Vaughan", price: 39.99}];`
 - Run block: `Run | Debug Try | Visualize`
 - Service function: `resource function get albums() returns Album[] { return albums.toArray(); }`
 - Resource function: `resource function get albums(string id) returns Album|http:NotFound { Album album = albums[id]; if (album is null) { return http:NOT_FOUND; } else { return album; } }`
 - Resource function: `resource function post albums(@httpPayload Album album) returns Album { albums.addVal(album); return album; }`
 - Resource function: `resource function delete path(string id)(string? param) returns error?|http:NotFound|http:Accepted { Album? album = albums[id]; if (album is null) { return http:NOT_FOUND; } else { _ = albums.remove(id); return http:ACCEPTED; } }`
 - Overview Diagram (restapi):** A diagram showing the service structure with components and their relationships.
 - Configure Resource (restapi):** A dialog for configuring a resource. It shows:
 - HTTP Method: GET
 - Resource Path: `/albums`
 - Parameters: None
 - Advanced Parameters: Show
 - Responses: 500 error?
 - Bottom Status Bar:** Shows the current version of the IDE: `Ballerina 2201.7.2 (Swan Lake Update 7)`.

Use Case 5: Visual Features

Data mapping

The screenshot shows the Ballerina IDE interface with two main panes. On the left, the code editor displays the file `main.bal` containing Ballerina code for handling calendar events. On the right, a Data Mapper diagram titled "Data Mapper: calEventToTrelloCard" shows the mapping between a `calEvent` object and a `trelloCards` object.

Code Editor (main.bal):

```
main.bal Diagram — Integration-samples
main.bal x
Run | Debug | Try it
service calendar:CalendarService on calendarListener {
    remote function onNewEvent(calendar:Event payload) returns error?
    do {
        // Add the card to the Trello list
        trello:Cards card = calEventToTrelloCard(payload);
        _ = check trello->addCards(card);

        // Send SMS notification
        string twilioMsg = calEventToMessage(payload);
        _ = check twilio->sendSms(twFromMobile, twToMobile, twMsg);
    } on fail var e {
        // Log the error and add the event to the dead letter channel
        log:printErrorString "Failed to process the calendar event";
        toDeadLetterChannel(payload, e);
    }
}

remote function onEventDelete(calendar:Event payload) returns error?
{
}

remote function onEventUpdate(calendar:Event payload) returns error?
{
}
```

Data Mapper Diagram:

The Data Mapper diagram illustrates the mapping between the `calEvent` and `trelloCards` objects. The `calEvent` object has properties like `kind`, `etag`, `id`, `status`, `htmlLink`, `created`, `updated`, `summary`, `description`, `location`, `colorId`, `creator`, `organizer`, `start`, `end`, and `end`. The `trelloCards` object has properties like `closed`, `desc`, `due`, `fileSource`, `idAttachmentCover`, `idBoard`, `idCardSource`, `idLabel`, `idList`, `idMembers`, `keepFromSource`, `labels`, `name`, `pos`, `subscribed`, and `urlSource`. Blue curved arrows indicate the flow of data from specific fields in the `calEvent` object to corresponding fields in the `trelloCards` object.

Use Case 5: Visual Features

Data persistence

The screenshot shows the Ballerina IDE interface with two main panes.

Code Editor: The left pane displays the `model.bal` file content, which defines three record types: User, Contest, and Challenge. The User record includes fields like id, username, fullname, role, challenge, moderatedContests, and submissions. The Contest record includes fields like id, title, readmeFile, startTime, endTime, imageUrl, challenges, submissions, and moderator. The Challenge record includes fields like id, title, createdTime, templateFile, readmeFile, difficulty, testCasesFile, contests, author, and submissions.

```
entity_model > persist > model.bal > Submission >
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 createdTime;
30     byte[] templateFile;
31     byte[] readmeFile;
32     string difficulty;
33     byte[] testCasesFile;
34     ChallengesOnContests[] contests;
35     User author;
36     Submission[] submissions;
37 };
```

Entity Relationship Diagram: The right pane shows the Entity Relationship Diagram (ERD) titled "Entity Relationship Diagram — ballroom-backend". It illustrates the relationships between four entities: Contest, Challenge, User, and Submission. The `ChallengesOnContests` association connects Contest and Challenge. The `ContestsOnContests` association connects Challenge and Contest. The `ContestsOnContests` association connects User and Contest. The `ContestsOnContests` association connects Submission and Contest. The `ContestsOnContests` association connects User and Submission. The `ContestsOnContests` association connects Submission and User.

```
graph TD
    Contest -- "0..>|ChallengesOnContests|---" Challenge
    Challenge -- "0..>|ContestsOnContests|---" Contest
    User -- "1..>|ContestsOnContests|---" Contest
    User -- "1..>|ContestsOnContests|---" Submission
    Submission -- "0..>|ContestsOnContests|---" Contest
    Submission -- "0..>|ContestsOnContests|---" User
```

Use Case 5: Visual Features

Text and graphical syntax parity

The screenshot shows the Ballerina IDE interface with two main panes. The left pane displays the Ballerina code for a 'main.bal' file, and the right pane displays a corresponding state transition diagram.

Code (left pane):

```
github-pull-requests-to-gsheets > E main.bal > ⌂ PR > ↗ url
1 import ballerina/http;
2 import ballerina/gcp/gsheet.sheets;
3
4 configurable string githubPAT = "?";
5 configurable string sheetsAccessToken = "?";
6 configurable string spreadSheetId = "?";
7 configurable string sheetName = "Sheet1";
8
9 type PR record {
10     string url;
11     string title;
12     string state;
13     string created_at;
14     string updated_at;
15 };
16
17 public function main() returns error? {
18     http:Client github = check new ("https://api.github.com");
19     map<string> headers = {
20         "Accept": "application/vnd.github.v3+json",
21         "Authorization": "token " + githubPAT
22     };
23
24     // Network data == program data
25     PR[] prs = check github->repos/octocat>Hello-World/pulls(headers);
26
27     sheets:Client gsheets = check new ((auth: {Token: sheetsAccessToken}));
28     check gsheets->appendRowToSheet(spreadSheetId, sheetName,
29         ["Issue", "Title", "State", "Created At", "Updated At"]);
30
31     foreach var [url, title, state, created_at, updated_at] in prs {
32         check gsheets->appendRowToSheet(spreadSheetId, sheetName,
33             [url, title, state, created_at, updated_at]);
34     }
35 }
36
```

Diagram (right pane):

```
graph TD
    START((START)) -- new --> github((github))
    github -- get --> PRS((PR[]))
    PRS -- new --> gsheets((gsheets))
    gsheets -- appendRowToSheet --> END((END))
    END -- feedback loop --> PRS
```

The diagram illustrates the flow of data from GitHub to Google Sheets. It starts with a 'START' state, followed by a 'new' action leading to a 'github' state. From the 'github' state, a 'get' action leads to a 'PRS' state (represented as a rectangle containing 'PR[]'). From the 'PRS' state, a 'new' action leads to a 'gsheets' state. Finally, a 'appendRowToSheet' action leads to an 'END' state. A feedback loop returns from the 'END' state back to the 'PRS' state.

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?

