WSO2CON 2025

Natural Programming

Maryam Ziyad
Senior Technical Lead
WSO2

Programming

Writes and understands

Programming Language

Understands and executes

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

- Donald Knuth -

Literate programming

- Writing code in a way that is human-readable and understandable, intertwining natural language explanations with the code itself.
- Natural language is still for humans (developers).

Literate programming example (HWTIME – Knuth, 1992)

```
\datethis
@* Extended Hello World program.
This is a medium-short demonstration of \.{CWEB}.

@c
@<Include files@>@;
@<Global variables@>@;
@<Subroutines@>@;
main()
{
  @<Local variables@>;
  @<Print a greeting@>;
  @<Print the date and time@>;
  @<Print an interesting date and time
    from the past@>;
}

@ First we brush up our Shakespeare by quoting
from The Merchant of Venice (Act~I, Scene~I,
Line~77). This makes the program literate.

@<Print a greeting@>=
printf("Greetings ... to\n"); /* Hello, */
printf(" `the stage, ");       /* world */
printf("where every man must play a part'.\n");

@ Since we're using the |printf| routine, we had
better include the standard input/output header
file.

@<Include files@>=
#include <stdio.h>

@ Now we brush up our knowledge of \Cee\ runtime
routines, by recalling that the function |time(0)|
returns the current time in seconds.

@<Print the date and time@>=
current_time=time(0);
printf("Today is ");
print_date(current_time);
printf(",\n and the time is ");
print_time(current_time);
printf(".\n");
```

Literate programming

Writes and understands

Natural Language

describes

Programming Language

Understands and executes

The hottest new programming language is English

- Andrej Karpathy -

Programming today

- With the emergence of/advances in generative AI, LLMs, and Copilots, use of natural language in programming has changed significantly
  - Not just at development time, but also at runtime
- Natural language is now a more significant part of programming and coding artifacts
- Encourages increased participation in programming and software development
  - More and more non-developer/non-technical users

Natural programming

We see an evolution of programming, where the process of programming is one of **co-creation** with computers, where both human and machine express intent and actions in both natural language and in programming languages

Natural programming

Writes and understands

Writes and understands

Natural Language

+

Programming Language

Writes, understands, and executes

Natural Language **+ Programming** Language

=

Natural Programming

A blend of natural language and programming language

- Natural language is ambiguous, programming language is not
- Not suggesting replacing code entirely with natural language
- Instead,
  - leverage natural language where it can enable new functionality
  - simplify integrating natural language into programming logic
- Blend natural language and programming language to maximize value

Reference implementation in Ballerina

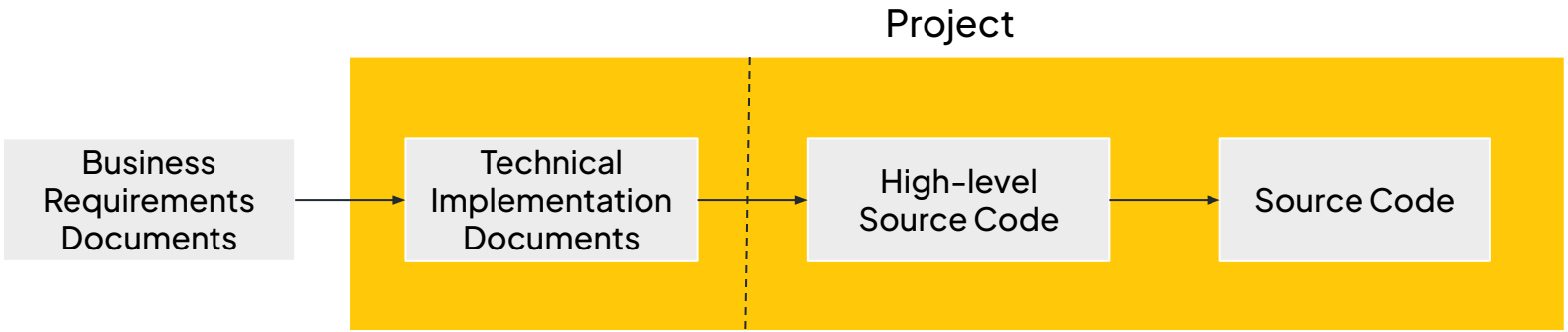
- Most of what we are proposing as natural programming can be applied to most programming languages
- Working on a reference implementation in Ballerina
  - An open-source, cloud-native programming language optimized for integration
  - Developed by WSO2

Natural programming in the software development process and code

- Development time
- Compile time
- Runtime

Development time

Traditional software development process



Development time – Bringing requirements into the project

- Bridging the gap between business users and the implementation, by making business user requirements become part of the project.
- These requirements can be used to
  - Generate code
  - Generate tests
  - Check for drift between requirements and the implementation

Scenario

Claim management platform

Overview

Employees of the organization (users) may need to spend on work-related expenses for which they can subsequently raise claims. The requirement is to implement a claims management platform that would enable managing these claims online.

The following functionality should be supported.

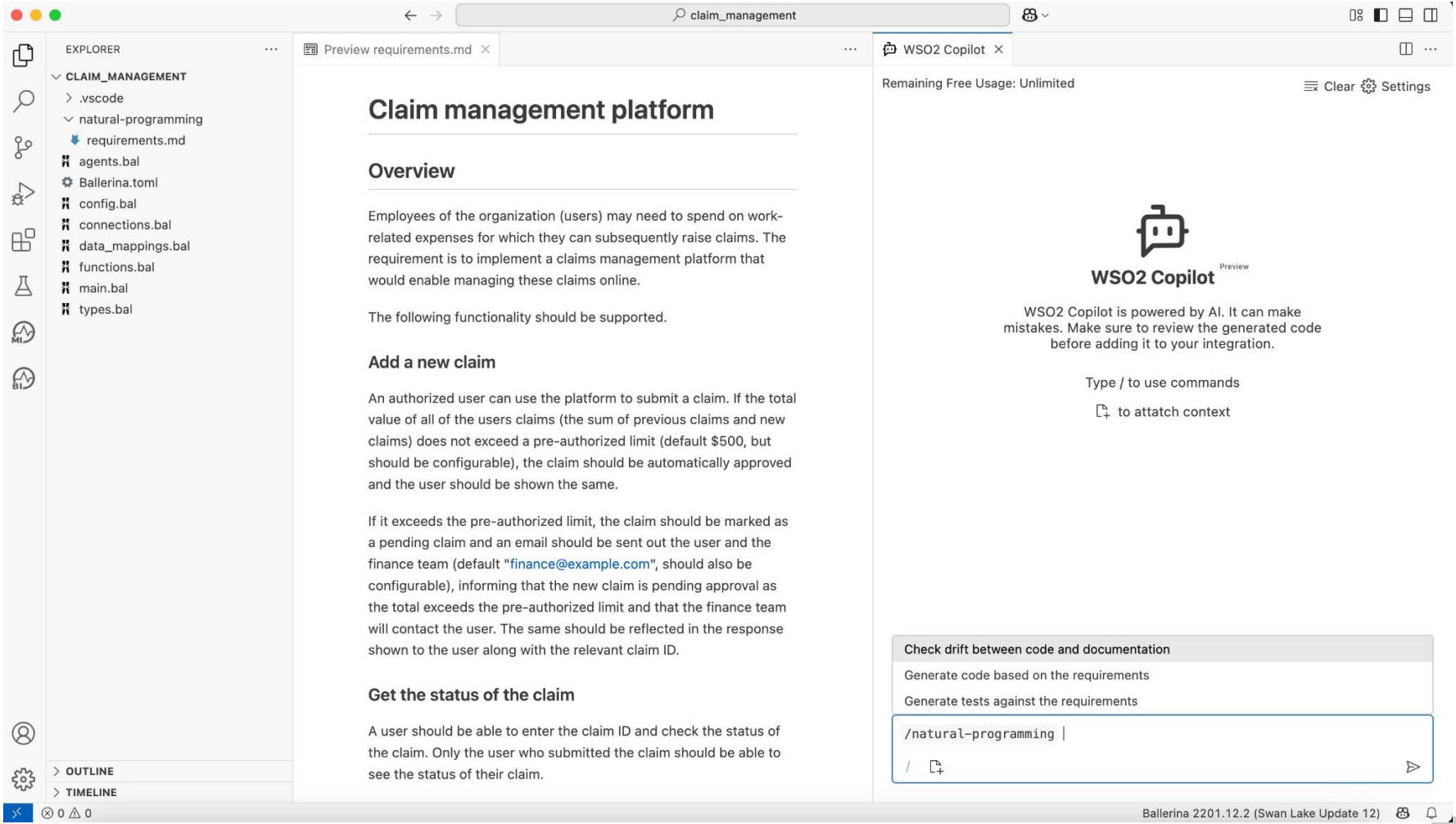
Add a new claim

An authorized user can use the platform to submit a claim. If the total value of all of the users claims (the sum of previous claims and new claims) does not exceed a pre-authorized limit (default $500, but should be configurable), the claim should be automatically approved and the user should be shown the same.

If it exceeds the pre-authorized limit, the claim should be marked as a pending claim and an email should be sent out the user and the finance team (default "finance@example.com", should also be configurable), informing that the new claim is pending approval as the total exceeds the pre-authorized limit and that the finance team will contact the user. The same should be reflected in the response shown to the user along with the relevant claim ID.

Get the status of the claim

A user should be able to enter the claim ID and check the status of the claim. Only the user who submitted the claim should be able to see the status of their claim.


claim_management
EXPLORER
CLAIM_MANAGEMENT
.vscode
natural-programming
requirements.md
agents.bal
Ballerina.toml
config.bal
connections.bal
data_mappings.bal
functions.bal
main.bal
types.bal
OUTLINE
TIMELINE
Preview requirements.md
Claim management platform
Overview
Employees of the organization (users) may need to spend on work-related expenses for which they can subsequently raise claims. The requirement is to implement a claims management platform that would enable managing these claims online.
The following functionality should be supported.
Add a new claim
An authorized user can use the platform to submit a claim. If the total value of all of the users claims (the sum of previous claims and new claims) does not exceed a pre-authorized limit (default $500, but should be configurable), the claim should be automatically approved and the user should be shown the same.
If it exceeds the pre-authorized limit, the claim should be marked as a pending claim and an email should be sent out the user and the finance team (default "finance@example.com", should also be configurable), informing that the new claim is pending approval as the total exceeds the pre-authorized limit and that the finance team will contact the user. The same should be reflected in the response shown to the user along with the relevant claim ID.
Get the status of the claim
A user should be able to enter the claim ID and check the status of the claim. Only the user who submitted the claim should be able to see the status of their claim.
WSO2 Copilot
Remaining Free Usage: Unlimited
Clear
Settings
WSO2 Copilot
Preview
WSO2 Copilot is powered by AI. It can make mistakes. Make sure to review the generated code before adding it to your integration.
Type / to use commands
to attatch context
Check drift between code and documentation
Generate code based on the requirements
Generate tests against the requirements
/natural-programming
Ballerina 2201.12.2 (Swan Lake Update 12)

Development time – Code generation

- Similar to code generation with a Copilot
- Aware that requirements can be at a higher (business user) level, and that there would generally be developer input too
- Encourages proper API documentation

Development time – Test generation

- Generate tests based on the requirements, against the implementation
  - Uses just the APIs from the implementation
- Can be considered a form of user acceptance testing
- Complements tests written/generated by a developer

Development time – Drift check

- Checks for drift between business requirements, documentation, and the implementation
- Backed by LLMs, runs periodically in the background if enabled or can run on demand. Also suggest changes to the code or the documentation to remove/minimize drift.
- Encourages proper API and project documentation
- Makes code easier to understand and evolve, while staying in sync with the requirements

Development time – Drift check

```
    # Sends an email notification for pending claims
    # + userEmail - Email address of the user
    # + financeEmail - Email address of the finance team
    # + claimId - ID of the claim
    # + return - Error if email sending fails
    Visualize
```

The requirement specifies that when a claim exceeds the limit, email should be sent to both user and finance team. However, the code only sends email to the user. (NLE001)

View Problem (⌥F8) Quick Fix... (⌘.) Fix using Copilot (⌘I)

```
? {
e pre-author
            to: [userEmail],
            subject: "Claim Pending Approval",
            body: emailBody
        };

        check smtpClient->sendMessage(emailMessage);
    }
}
```

Development time – Drift check

```
private function sendPendingClaimEmail(string userEmail, string financeEmail, string claimId) returns error? {
    string emailBody = string `Your claim (ID: ${claimId}) has been submitted for approval as it exceeds the pre-a

    email:Message emailMessage = {
        to: [userEmail],
        subject: "Claim Pending Approval",
        body: emailBody
    };

                                                    ssage(emailMessage);
```

Quick Fix

Update code to match docs

Fix using Copilot

Development time – Drift check

```
private function sendPendingClaimEmail(string userEmail, string financ
    string emailBody = string `Your claim (ID: ${claimId}) has been su

    email:Message emailMessage = {
        to: [userEmail],
        subject: "Claim Pending Approval",
        body: emailBody
    };


    check smtpClient->sendMessage(emailMessage);
}
```

```
107     private function sendPendingClaimEmail(string userEmail, string fir
108         string emailBody = string `Your claim (ID: ${claimId}) has beer
109
110+ email:Message emailMessage = {
111+     to: [userEmail, financeEmail],
112+     subject: "Claim Pending Approval",
113+     body: emailBody
114+ };
115
116
117         check smtpClient->sendMessage(emailMessage);
118     }
```

Development time – Drift check

- A common problem with natural language in programming artifacts (e.g., comments in the source code and project documentation traditionally) is that they can go out of sync with the code.
- Drift check can also address this problem for all kinds of documentation – from comments, to API documents, to other project documentation.

Development time – Preserving user intent

- Use of AI models for code generation continues to increase
- Usually an interactive/iterative process between user and AI assistant
- Prompts are generally discarded once the user is satisfied with the generated code, but the prompts can contain useful contextual information that doesn't get reflected entirely in the code.
- Another form of natural language input that would be useful to preserve in some form

Development time – Preserving user intent

- With natural programming, we preserve a summary of the user intentions (via prompts) that led to the specific version of the code, where the code is AI generated
    - Added in a developer.md file
- Not a history of user interactions, but an effective summary

Development time – Preserving user intent – Sample

Claim management implementation, but assume the implementation was generated

- maintaining claim data in memory
- sending two separate emails to the user and the finance team

Development time – Preserving user intent – Sample prompts

> Use a MySQL database to persist data.

> Can we have two tables in the database for claims and total_claims against user id? Have the total_claims table be updated via a database trigger.

> Send a single email to both the employee and the finance team instead of two separate emails.

> Use MS SQL instead of MySQL.

Development time – Preserving user intent – Sample summary

Use MS SQL for data persistence with two tables: claims and total_claims. The total_claims table should be updated via a database trigger based on the user id. Send a single email to both the employee and the finance team.

Natural language in code

Natural language usage

- Natural language usage so far (development time) has still most often been outside the code
  - Except for API documentation and comments
- What if we can interleave natural language and code?
  - Compile time
  - Runtime

Compile time

Natural language interleaved with code

- Specify the implementation of a function in natural language in the code
- The compiler generates code based on the requirements
  - Copilot for the compiler?
- Potentially the closest parallels with literate programming

population-data > main.bal > ...

```
1
2   import ballerina/http;
3   import ballerina/io;
4   import ballerinax/np;
5
6   configurable string populationDataUrl = ?;
7
8   type Country record {
9       string country;
10      string continent;
11      int population;
12      decimal gdp;
13  };
14
15  type GDPPerCapita record {|
16      string country;
17      string continent;
18      decimal gdpPerCapita;
19  |};
20
21  function getCountriesWithHighestGdpPerCapitaInContinent(Country[] countries, string continent)
22      returns GDPPerCapita[]|error = @np:GenerateCode {
23          prompt: string `Filter the 10 countries with a population of at least 10,000,000 with
24                          the highest GDP per capita in the given continent and return the data.`
25      } external;
26
27  public function main(string continent) returns error? {
28      http:Client populationEp = check new (populationDataUrl);
29      Country[] countries = check populationEp->/countries;
30
31      GDPPerCapita[] summary =
32          check getCountriesWithHighestGdpPerCapitaInContinent(countries, continent);
33      io:println(summary);
34  }
```

Natural language interleaved with code

- Potential to increase readability and convey what the implementation does in natural language, but no human in the loop!
- Works best with granular, unambiguous requirements – user chooses when to use
- The implementation could become a bit of a black box – developers may generally tend to prefer generating code at development time.
    - Ballerina adds the generated code into the project at compile time - a user can still look at the generated code after building, before running

Generated code

population-data > generated > getCountriesWithHighestGdpPerCapitaInContinent_NPGenerated.bal > ...

```
1
2   function getCountriesWithHighestGdpPerCapitaInContinent_NPGenerated(
3           Country[] countries, string continent) returns GDPPerCapita[]|error {
4       return from Country country in countries
5           where country.continent == continent && country.population >= 10000000
6           let decimal gdpPerCapita = country.gdp / (<decimal>country.population)
7           order by gdpPerCapita descending
8           limit 10
9           select {
10              country: country.country,
11              continent: country.continent,
12              gdpPerCapita: gdpPerCapita
13          };
14  }
15
```

Natural language interleaved with code

- Currently experimental
  - The generated code can change with each build - requires further consideration for maintenance, versioning, etc.
  - No human in the loop to review and accept the generated code, which could open room for issues including security concerns. Initial version enforces some restrictions compared to a traditional Copilot to avoid unintentional behaviour.

Runtime

Natural language instructions at runtime

- Many languages (and frameworks), including Ballerina, support calling LLMs as a part of the program logic
- Allows expanding the scope of problems that can be solved
- Allows leveraging the strengths of the programming language and capabilities enabled via an LLM to achieve subtasks that might not cater to the language's strengths
- The user decides what can be implemented in code vs what needs to be implemented using an LLM

Natural language instructions at runtime

- Calling an LLM in the code, generally includes
  - Initializing a client that can talk to the LLM
  - Specifying an expected format for responses
  - Parsing content from the response to a user-defined type
- Requires quite a bit of boilerplate code that could take the focus away from the core task described in natural language

Blog site backend service

- Submit a new post
  - LLM call to identify the most suitable category (from a specified list) and a rating out of 1 - 10 (based on a defined criteria) - requirement specified in natural language
  - If there is a suitable category and the rating is greater than 3, accept the submission and persist the data. If not, reject the post - implemented entirely in the programming language.
- Retrieve posts by category, sorted by a rating identified when the blog post is created - implemented entirely in the programming language.

Blog site backend service - Overview

Ballerina JSON to type mapping

```json
{
    "id": "A1212456",
    "name": "John Doe",
    "age": 30,
    "email": "johndoe@example.com",
    "address": {
        "street": "1234 Main St",
        "city": "Springfield",
        "state": "IL",
        "zip": "62701"
    }
}
```

```
type Employee record {|
   string id;
   string name;
   int age;
   string email;
   Address address;
|};

type Address record {|
   string street;
   string city;
   string state;
   string zip;
|};
```

Blog site backend service – Types

```
# Represents a blog entry with title and content.
public type Blog record {|
    # The title of the blog
    string title;
    # The content of the blog
    string content;
|};

# Review of a blog entry.
type Review record {|
    # Suggested category
    string? suggestedCategory;
    # Rating out of 10
    int rating;
|};
```

Blog site backend service – Client initialization

```
import ballerinax/azure.openai.chat;

configurable string apiKey = ?;
configurable string serviceUrl = ?;
configurable string deploymentId = ?;

final chat:Client chatClient = check new (
    config = {auth: {apiKey: apiKey}},
    serviceUrl = serviceUrl
);
```

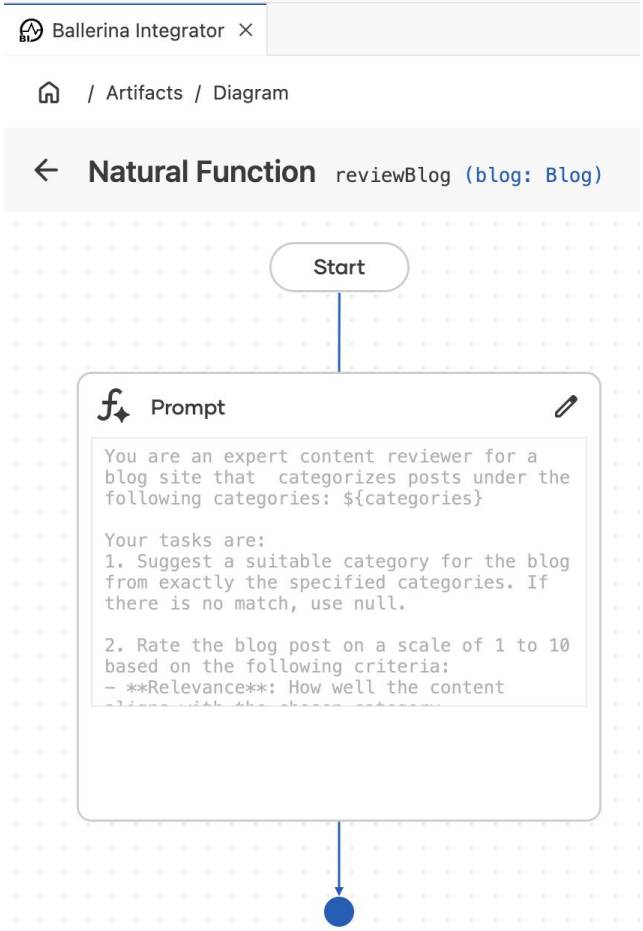
Blog site backend service - using the LLM Client

```
final readonly & string[] categories = [
    "Tech Innovations & Software Development",
    "Programming Languages & Frameworks",
    "DevOps, Cloud Computing & Automation",
    "Career Growth in Tech",
    "Open Source & Community-Driven Development"
];
```

```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        string prompt = string `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories.toString()}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Provide your response in the following format:
            {
                "suggestedCategory": "Category Name" (or null if no category fits),
                "rating": 7 (replace with the appropriate rating)
            }

            Here is the blog post content:

            Title: ${blog.title}
            Content: ${blog.content}`;

        chat:CreateChatCompletionRequest chatBody = {
            messages: [{role: "user", "content": prompt}]
        };

        chat:CreateChatCompletionResponse chatResult = check chatClient->/deployments/[deploymentId]/ch
        record {|
            chat:ChatCompletionResponseMessage message?;
            chat:ContentFilterChoiceResults content_filter_results?;
            int index?;
            string finish_reason?;
            anydata...;
        |}[] choices = check chatResult.choices.ensureType();

        string resp = check choices[0].message?.content.ensureType();

        string processedResponse = re `${"```json|```"}`.replaceAll(resp, "");

        Review {suggestedCategory, rating} = check processedResponse.fromJsonStringWithType();
```

Natural functions

- A special kind of function
  - Uses a prompt associated with the function in a call to an LLM
  - The JSON schema for the expected response is generated automatically from the type
  - The relevant content is extracted from the response and is parsed as the type expected by the user


Ballerina Integrator
/ Artifacts / Diagram
Natural Function reviewBlog (blog: Blog)
Start
Prompt
You are an expert content reviewer for a blog site that categorizes posts under the following categories: ${categories}
Your tasks are:
1. Suggest a suitable category for the blog from exactly the specified categories. If there is no match, use null.
2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
- **Relevance**: How well the content

Blog site backend service – using natural functions

```
public isolated function reviewBlog(
    Blog blog,
    np:Prompt prompt = `You are an expert content reviewer for a blog site that
        categorizes posts under the following categories: ${categories}

        Your tasks are:
        1. Suggest a suitable category for the blog from exactly the specified categories.
           If there is no match, use null.

        2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
        - **Relevance**: How well the content aligns with the chosen category.
        - **Depth**: The level of detail and insight in the content.
        - **Clarity**: How easy it is to read and understand.
        - **Originality**: Whether the content introduces fresh perspectives or ideas.
        - **Language Quality**: Grammar, spelling, and overall writing quality.

        Here is the blog post content:

        Title: ${blog.title}
        Content: ${blog.content}`) returns Review|error = @np:NaturalFunction external;

resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        Review {suggestedCategory, rating} = check reviewBlog(blog);
```

Natural functions

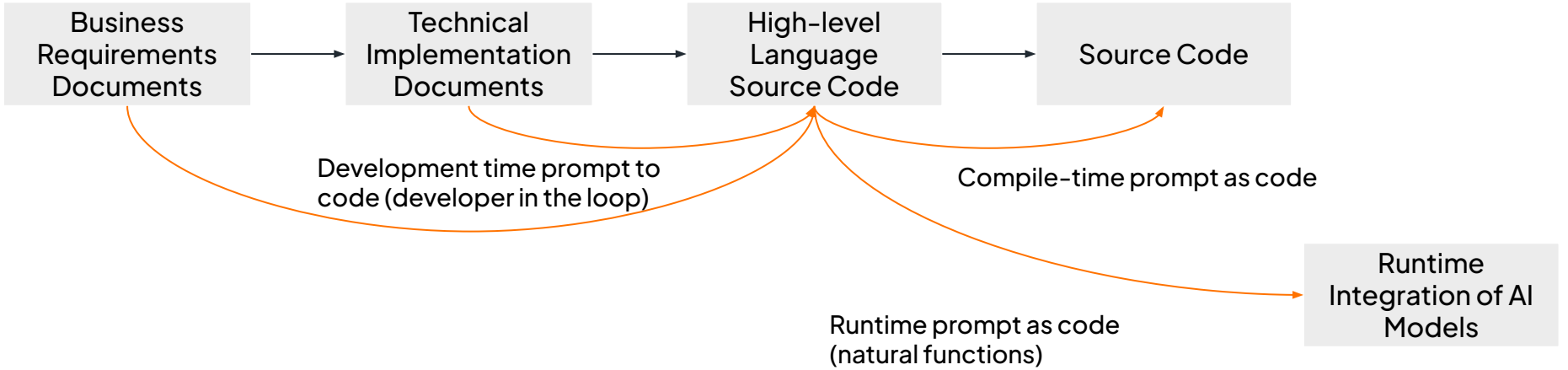
- Easier to read and understand
- Easier to work with the response
- Reduces the amount of code the user has to write, by abstracting out the common logic
    - Model is configured via configurable variables - applies to the entire project
    - Alternatively, can introduce a parameter to the function where function-wise control is required
- Ensures schema and expected type don’t go out of sync

```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        string prompt = string `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories.toString()}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Provide your response in the following format:
            {
                "suggestedCategory": "Category Name" (or null if no category fits),
                "rating": 7 (replace with the appropriate rating)
            }

            Here is the blog post content:

            Title: ${blog.title}
            Content: ${blog.content}`;

        chat:CreateChatCompletionRequest chatBody = {
            messages: [{role: "user", "content": prompt}]
        };

        chat:CreateChatCompletionResponse chatResult = check chatClient->/deployments/[deploymentId]/ch
        record {|
            chat:ChatCompletionResponseMessage message?;
            chat:ContentFilterChoiceResults content_filter_results?;
            int index?;
            string finish_reason?;
            anydata...;
        |}[] choices = check chatResult.choices.ensureType();

        string resp = check choices[0].message?.content.ensureType();

        string processedResponse = re `${"```json|```"}`.replaceAll(resp, "");

        Review {suggestedCategory, rating} = check processedResponse.fromJsonStringWithType();
```

```
public isolated function reviewBlog(
    Blog blog,
    np:Prompt prompt = `You are an expert content reviewer for a blog site that
        categorizes posts under the following categories: ${categories}

        Your tasks are:
        1. Suggest a suitable category for the blog from exactly the specified categories.
           If there is no match, use null.

        2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
        - **Relevance**: How well the content aligns with the chosen category.
        - **Depth**: The level of detail and insight in the content.
        - **Clarity**: How easy it is to read and understand.
        - **Originality**: Whether the content introduces fresh perspectives or ideas.
        - **Language Quality**: Grammar, spelling, and overall writing quality.

        Here is the blog post content:

        Title: ${blog.title}
        Content: ${blog.content}`) returns Review|error = @np:NaturalFunction external;
```

```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        Review {suggestedCategory, rating} = check reviewBlog(blog);
```

Reimagining the software development process



Current status

- Early days!
- Current versions of Ballerina and the Ballerina Integrator support natural functions and the development-time features
    - Compile-time prompt as code is still experimental

(Immediate) Future work

- Making natural functions even more first-class - make the natural language instructions the body of a function.
- Exploring extending natural functions to support tool calling
- Support for multi-modal requirements
- Scaling natural programming to handle larger-scale applications

Question Time!


Thank you!
WSO2CON2025