



Agent and Tools

The core premise of agents involves employing a language model to discern a sequence of actions. Unlike *chains*, where actions are hardcoded, agents utilize a language model as a reasoning engine to determine the actions (*tools*) and their sequence.

Concepts

Agent

The agent is the core decision-maker responsible for determining the subsequent steps. It's driven by a language model and a prompt. The inputs include:

- Tools: Descriptions of available actions
- User Input: The high-level objective
- Intermediate Steps: Previous (action, tool output) pairs executed in the order to fulfill the user input

The output presents the next action(s) to take or the final response to be delivered to the user. Each action specifies a tool (method name) and its corresponding input parameters.

Tools

Tools are the methods an agent can invoke. Two elements are essential when using tools:

- Granting the agent access to the right tools.
- Describing tools in a manner most beneficial to the agent.

Failure to consider both aspects may hinder the agent's functionality. Access to an incorrect set of tools impedes the agent from achieving objectives, while improperly described tools hamper their effective utilization.

Recommendations for Agent Construction

When constructing an agent, consider:

- Setting the model temperature to 0 for the agent to consistently choose the most probable action.

- Ensuring well-detailed descriptions of tools.
- Listing steps in the prompt in the desired execution order.

Declaring a Tool

A *tool* denotes a method accessible by an agent. It must be part of a CDI bean and have at least one method annotated with `@Tool`:

```
// Example of an EmailService tool method
package io.quarkiverse.langchain4j.sample;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;

import dev.langchain4j.agent.tool.Tool;
import io.quarkus.logging.Log;
import io.quarkus.mailer.Mail;
import io.quarkus.mailer.Mailer;

@ApplicationScoped
public class EmailService {

    @Inject
    Mailer mailer;

    @Tool("send the provided content via email")
    public void sendAnEmail(String content) {
        Log.info("Sending an email: " + content);
        mailer.send(Mail.withText("sendMeALetter@quarkus.io", "A poem for you",
content));
    }

}
```

IMPORTANT

A CDI bean can host multiple methods annotated with `@Tool`. However, ensure each method name is unique among all declared tools.

Providing Tool Access

In the [AI Service](#), you can specify the tools accessible to the agent. By default, no tools are available. Hence, ensure to define the list of tools you wish to make accessible:

```
@RegisterAiService(tools = EmailService.class)
public interface MyAiService {
    // ...
}
```

When employing tools, configuring the memory provider is necessary. Since tools necessitate a sequence of messages, maintaining this conversation is essential. A minimum memory of three messages is necessary for optimal tool functionality.

How do tools work?

The question of how tools work naturally arises given the fact that no code needs to be written that wires up their usage.

The short answer is by providing the proper user and system messages and tool descriptions, the extension is able to craft API requests that allow the LLM to respond back with the proper tool invocations - but it is the extension itself then proceeds to invoke the tool. The result of the invocation is then used by the extension to make another request to the LLM which will then response with a new answer, potentially leading to another round of a tool invocation and the crafting of a new request to the LLM.

Detailed example

To help clarify the various interactions, let's use the following example where we are creating a very simple calculator chatbot that responds to user requests over HTTP, leveraging the `quarkus-langchain4j-openai` extension. Here is the full source code:

```
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import jakarta.annotation.PreDestroy;
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Singleton;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;

import org.jboss.resteasy.reactive.RestQuery;

import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import io.quarkiverse.langchain4j.RegisterAiService;
import io.quarkiverse.langchain4j.RemovableChatMemoryProvider;
```

```
@Path("assistant-with-tool")
public class AssistantWithToolsResource {

    private final Assistant assistant;

    public AssistantWithToolsResource(Assistant assistant) {
        this.assistant = assistant;
    }

    @GET ③
    public String get(@RestQuery String message) {
        return assistant.chat(message);
    }

    @Singleton ①
    public static class Calculator {

        @Tool("Calculates the length of a string")
        int stringLength(String s) {
            return s.length();
        }

        @Tool("Calculates the sum of two numbers")
        int add(int a, int b) {
            return a + b;
        }

        @Tool("Calculates the square root of a number")
        double sqrt(int x) {
            return Math.sqrt(x);
        }
    }

    @RegisterAiService(tools = Calculator.class) ②
    public interface Assistant {

        String chat(String userMessage);
    }
}
```

- ① Declare a CDI bean that provides three different tools
- ② Register an AiService that responds to a user's request and has access to the calculator tools. This service is also able to keep track of the session's messages using the CDI message store declared above.
- ③ Declare an HTTP endpoint that retrieves the user's question via a query parameter and simply responds with chatbot's response

Now, if we ask the chatbot What is the square root of the sum of the numbers of letters in the words "hello" and "world" via:

```
curl 'http://localhost:8080/assistant-with-tool?
message=What%20is%20the%20square%20root%20of%20the%20sum%20of%20the%20numbers%20of%
20letters%20in%20the%20words%20%22hello%22%20and%20%22world%22'
```

The response will be The square root of the sum of the numbers of letters in the words "hello" and "world" is approximately 3.162.

What is interesting however in understanding how the tools were used, is seeing the requests the extension has made to OpenAI and the responses the latter provides.

The application makes a first REST call to OpenAI that contains the following JSON payload:

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "What is the square root of the sum of the numbers of letters in
the words \"hello\" and \"world\""
    }
  ],
  "temperature": 1.0,
  "top_p": 1.0,
  "presence_penalty": 0.0,
  "frequency_penalty": 0.0,
  "functions": [
    {
      "name": "stringLength",
      "description": "Calculates the length of a string",
      "parameters": {
        "type": "object",
        "properties": {
          "s": {
            "type": "string"
          }
        }
      },
      "required": [
        "s"
      ]
    }
  ],
  {
    "name": "add",
```

```
{
  "description": "Calculates the sum of two numbers",
  "parameters": {
    "type": "object",
    "properties": {
      "a": {
        "type": "integer"
      },
      "b": {
        "type": "integer"
      }
    },
    "required": [
      "a",
      "b"
    ]
  }
},
{
  "name": "sqrt",
  "description": "Calculates the square root of a number",
  "parameters": {
    "type": "object",
    "properties": {
      "x": {
        "type": "integer"
      }
    },
    "required": [
      "x"
    ]
  }
}
]
```

There are two things to notice in this JSON payload:

- The user message which contains the user's question (the `messages` part of the request)
- The declaration of the three available functions (the `functions` part of the request) which directly map to the tools we declared in our source code.

The response from OpenAI is the following:

```
{
  "id": "chatcmpl-8L3uhmaMARLTbpGJGDvFW4pAPxM8V",
  "object": "chat.completion",
```

```
"created": 1700030623,
"model": "gpt-3.5-turbo-0613",
"choices": [
  {
    "index": 0,
    "message": {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "stringLength",
        "arguments": "{\n  \"s\": \"hello\"\n}"
      }
    },
    "finish_reason": "function_call"
  }
],
"usage": {
  "prompt_tokens": 118,
  "completion_tokens": 15,
  "total_tokens": 133
}
}
```

This response initially looks odd as it does not look like OpenAI has responded to user's question. However, upon closer inspection we see that the response contains something very interesting and that is the `function_call` part. In this part OpenAI is essentially telling the extension to invoke the `stringLength` tool with `hello` as the sole parameter. This is where the power of the LLM is evident - it has essentially figured out that to answer the user's question, it first needs to calculate the length of `"hello"`.

Upon receiving this response, the extension invokes `calculator.stringLength("hello")` and then sends the following new request to OpenAI:

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "What is the square root of the sum of the numbers of letters in the words \"hello\" and \"world\"?"
    },
    {
      "role": "assistant",
      "content": null,
    }
  ]
}
```

```
    "function_call": {
      "name": "stringLength",
      "arguments": "{\n  \"s\": \"hello\"\n}"
    },
    {
      "role": "function",
      "content": "5",
      "name": "stringLength"
    }
  ],
  "temperature": 1.0,
  "top_p": 1.0,
  "presence_penalty": 0.0,
  "frequency_penalty": 0.0,
  "functions": [
    {
      "name": "stringLength",
      "description": "Calculates the length of a string",
      "parameters": {
        "type": "object",
        "properties": {
          "s": {
            "type": "string"
          }
        }
      },
      "required": [
        "s"
      ]
    },
    {
      "name": "add",
      "description": "Calculates the sum of two numbers",
      "parameters": {
        "type": "object",
        "properties": {
          "a": {
            "type": "integer"
          },
          "b": {
            "type": "integer"
          }
        }
      },
      "required": [
        "a",
        "b"
      ]
    }
  ]
}
```



```
    }
  },
  {
    "name": "sqrt",
    "description": "Calculates the square root of a number",
    "parameters": {
      "type": "object",
      "properties": {
        "x": {
          "type": "integer"
        }
      },
      "required": [
        "x"
      ]
    }
  }
]
```

What is very important to notice here is that the extension is sending not only has the user's question, but also the LLM's first response requesting a tool invocation, along with a new message containing the result of the said invocation. Recall that LLMs are stateless, therefore the entire message history needs to be sent each time - this is the reason why the use of tools requires a chat memory.

OpenAI now responds with the following:

```
{
  "id": "chatcmpl-8L3ukBR1Pblqn4g5m3JocHQVzYqMy",
  "object": "chat.completion",
  "created": 1700030626,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": null,
        "function_call": {
          "name": "stringLength",
          "arguments": "{\n  \"s\": \"world\"\n}"
        }
      },
      "finish_reason": "function_call"
    }
  ]
}
```

```
],  
  "usage": {  
    "prompt_tokens": 142,  
    "completion_tokens": 15,  
    "total_tokens": 157  
  }  
}
```

Once again we see the LLM can not yet answer the original user's question, but instead instructs the extension to invoke a tool.

As you'd expect by now, the extension does just that and sends the following new request to OpenAI:

```
{  
  "model": "gpt-3.5-turbo",  
  "messages": [  
    {  
      "role": "user",  
      "content": "What is the square root of the sum of the numbers of letters in  
the words \"hello\" and \"world\""  
    },  
    {  
      "role": "assistant",  
      "content": null,  
      "function_call": {  
        "name": "stringLength",  
        "arguments": "{\n  \"s\": \"hello\"\n}"  
      }  
    },  
    {  
      "role": "function",  
      "content": "5",  
      "name": "stringLength"  
    },  
    {  
      "role": "assistant",  
      "content": null,  
      "function_call": {  
        "name": "stringLength",  
        "arguments": "{\n  \"s\": \"world\"\n}"  
      }  
    },  
    {  
      "role": "function",  
      "content": "5",
```

```
        "name": "stringLength"
    }
],
"temperature": 1.0,
"top_p": 1.0,
"presence_penalty": 0.0,
"frequency_penalty": 0.0,
"functions": [
    {
        "name": "stringLength",
        "description": "Calculates the length of a string",
        "parameters": {
            "type": "object",
            "properties": {
                "s": {
                    "type": "string"
                }
            }
        },
        "required": [
            "s"
        ]
    }
],
{
    "name": "add",
    "description": "Calculates the sum of two numbers",
    "parameters": {
        "type": "object",
        "properties": {
            "a": {
                "type": "integer"
            },
            "b": {
                "type": "integer"
            }
        },
        "required": [
            "a",
            "b"
        ]
    }
},
{
    "name": "sqrt",
    "description": "Calculates the square root of a number",
    "parameters": {
        "type": "object",
        "properties": {
```

```
        "x": {
          "type": "integer"
        },
        "required": [
          "x"
        ]
      }
    ]
  }
}
```

This new request, in addition to the previous one sent to OpenAI, contains the result of the invocation of `calculator.stringLength("world")`.

In turn, OpenAI responds with:

```
{
  "id": "chatcmpl-8L3uopZa8wdBmGWoWBqWbC08UdwVv",
  "object": "chat.completion",
  "created": 1700030630,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": null,
        "function_call": {
          "name": "add",
          "arguments": "{\\n  \\\"a\\\": 5,\\n  \\\"b\\\": 5\\n}"
        }
      },
      "finish_reason": "function_call"
    }
  ],
  "usage": {
    "prompt_tokens": 166,
    "completion_tokens": 21,
    "total_tokens": 187
  }
}
```

This time the LLM instructed the extension that it is time to invoke the `add` tool using `5` and `5` as arguments - these arguments of course are the results of the previous two `stringLength` invocations.

The extension upon receiving this response invoked `calculator.add(5,5)` and then crafts and sends the following message to OpenAI:

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "What is the square root of the sum of the numbers of letters in the words \"hello\" and \"world\"?"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "stringLength",
        "arguments": "{\n  \"s\": \"hello\"\n}"
      }
    },
    {
      "role": "function",
      "content": "5",
      "name": "stringLength"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "stringLength",
        "arguments": "{\n  \"s\": \"world\"\n}"
      }
    },
    {
      "role": "function",
      "content": "5",
      "name": "stringLength"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "add",
        "arguments": "{\n  \"a\": 5,\n  \"b\": 5\n}"
      }
    },
    {
      "role": "function",
```

```
        "content": "10",
        "name": "add"
    }
],
"temperature": 1.0,
"top_p": 1.0,
"presence_penalty": 0.0,
"frequency_penalty": 0.0,
"functions": [
    {
        "name": "stringLength",
        "description": "Calculates the length of a string",
        "parameters": {
            "type": "object",
            "properties": {
                "s": {
                    "type": "string"
                }
            },
            "required": [
                "s"
            ]
        }
    },
    {
        "name": "add",
        "description": "Calculates the sum of two numbers",
        "parameters": {
            "type": "object",
            "properties": {
                "a": {
                    "type": "integer"
                },
                "b": {
                    "type": "integer"
                }
            },
            "required": [
                "a",
                "b"
            ]
        }
    },
    {
        "name": "sqrt",
        "description": "Calculates the square root of a number",
        "parameters": {
            "type": "object",
```

```
    "properties": {
      "x": {
        "type": "integer"
      }
    },
    "required": [
      "x"
    ]
  }
}
```

The response from OpenAI is:

```
{
  "id": "chatcmpl-8L3urz1QWWjh3Dzmjqu003hK16NUE",
  "object": "chat.completion",
  "created": 1700030633,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": null,
        "function_call": {
          "name": "sqrt",
          "arguments": "{\\n  \\\"x\\\": 10\\n}"
        }
      },
      "finish_reason": "function_call"
    }
  ],
  "usage": {
    "prompt_tokens": 195,
    "completion_tokens": 14,
    "total_tokens": 209
  }
}
```

The LLM this time decided that the `sqrt` tool needs to be invoked.

The extension following this instruction invokes `calculator.sqrt(10)` (10 being the result of the previous `add` invocation) then crafts and sends the following message to OpenAI:

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "user",
      "content": "What is the square root of the sum of the numbers of letters in the words \"hello\" and \"world\"?"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "stringLength",
        "arguments": "{\n  \"s\": \"hello\"\n}"
      }
    },
    {
      "role": "function",
      "content": "5",
      "name": "stringLength"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "stringLength",
        "arguments": "{\n  \"s\": \"world\"\n}"
      }
    },
    {
      "role": "function",
      "content": "5",
      "name": "stringLength"
    },
    {
      "role": "assistant",
      "content": null,
      "function_call": {
        "name": "add",
        "arguments": "{\n  \"a\": 5,\n  \"b\": 5\n}"
      }
    },
    {
      "role": "function",
      "content": "10",
      "name": "add"
    },
  ],
}
```



```
{
  "role": "assistant",
  "content": null,
  "function_call": {
    "name": "sqrt",
    "arguments": "{\n  \"x\": 10\n}"
  }
},
{
  "role": "function",
  "content": "3.1622776601683795",
  "name": "sqrt"
}
],
"temperature": 1.0,
"top_p": 1.0,
"presence_penalty": 0.0,
"frequency_penalty": 0.0,
"functions": [
  {
    "name": "stringLength",
    "description": "Calculates the length of a string",
    "parameters": {
      "type": "object",
      "properties": {
        "s": {
          "type": "string"
        }
      }
    },
    "required": [
      "s"
    ]
  }
],
{
  "name": "add",
  "description": "Calculates the sum of two numbers",
  "parameters": {
    "type": "object",
    "properties": {
      "a": {
        "type": "integer"
      },
      "b": {
        "type": "integer"
      }
    },
    "required": [
```

```
        "a",
        "b"
    ]
},
{
    "name": "sqrt",
    "description": "Calculates the square root of a number",
    "parameters": {
        "type": "object",
        "properties": {
            "x": {
                "type": "integer"
            }
        },
        "required": [
            "x"
        ]
    }
}
]
```

Once again, it's very important to note that entire chat history has been sent by the extension to OpenAI. Using this entire chat history, the LLM is finally able to answer the original user's question:

```
{
  "id": "chatcmpl-8L3utMpnSKtgzWcmHJVwVJd36l9fk",
  "object": "chat.completion",
  "created": 1700030635,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "The square root of the sum of the numbers of letters in the words \"hello\" and \"world\" is approximately 3.162."
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 224,
    "completion_tokens": 29,
    "total_tokens": 253
  }
}
```

```
}  
}
```

As seen above, the answer is The square root of the sum of the numbers of letters in the words `"hello"` and `"world"` is approximately 3.162. and that is what Quarkus will send back to the user as the HTTP response.

Copyright (C) 2020-2024 [Red Hat](#) and individual contributors to [Quarkiverse](#).