Talk 5: How to develop a controller, extension, and application cases

Jiseong Kim
Ph.D. Student @ KAIST

Contact: jiseong@kaist.ac.kr

22, July, 2016

Contents

1. What is the **controller**? Is it **necessary**?
    1. What is the **key functions**?
1. Controller **API** & **Usage**
1. **Applications** with **Extension**
1. **Hands-on:** Harmonizing modules

What is “the controller”?
Is it “necessary”?

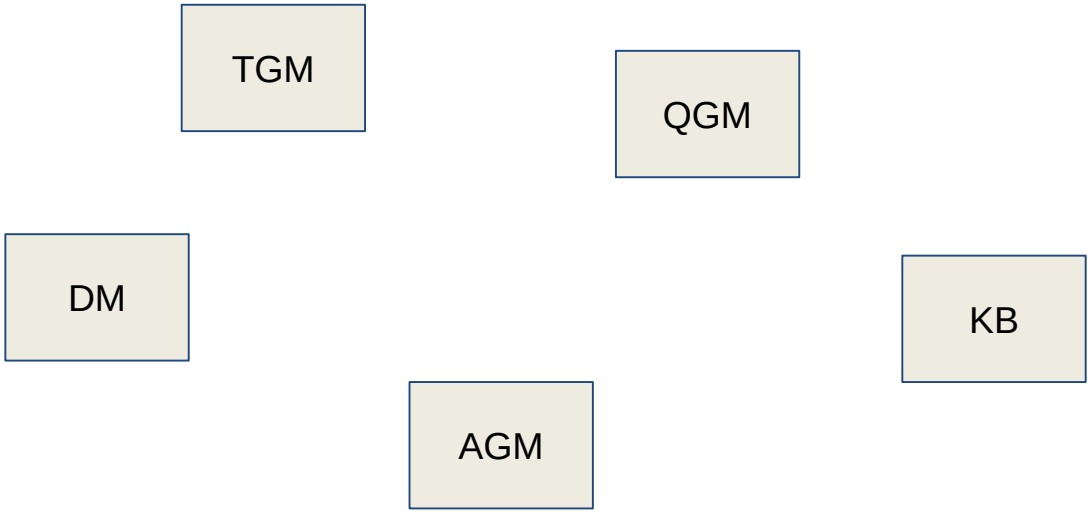
A Workflow Management System

- It provides an infrastructure for the **set-up**, **performance** and **monitoring** of a defined sequence of tasks, arranged as a workflow.

- It integrates separated tasks into a workflow.

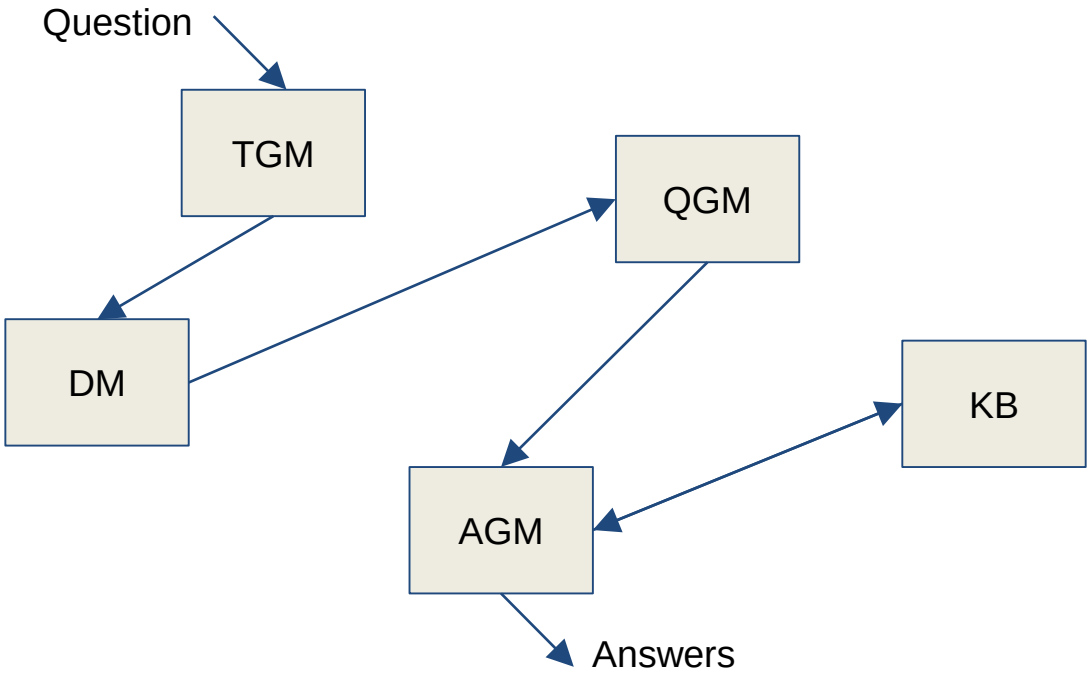
Workflow Manager for OKBQA

- Inherently, OKBQA modules are separated.
  - TGM, DM, QGM, AGM, KB, and so on



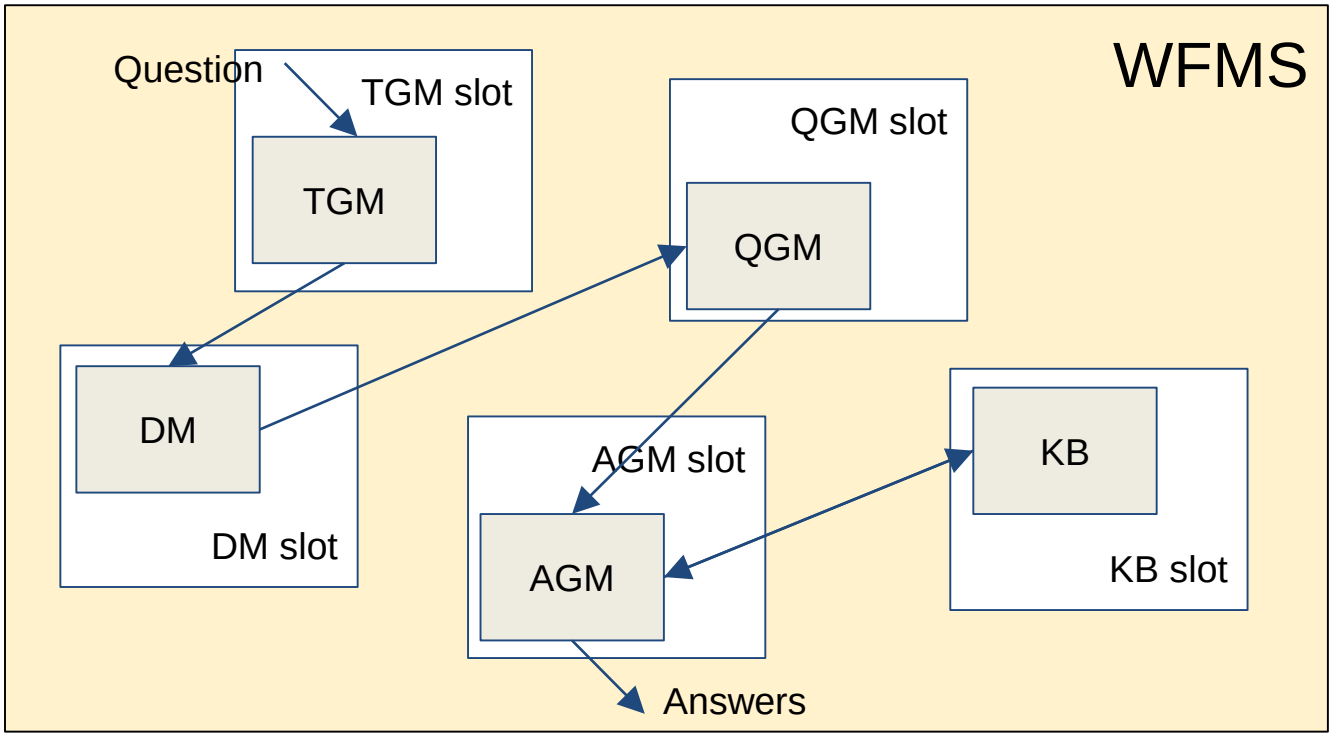
Workflow Manager for OKBQA

- They can be hard-wired.
  - Less flexibility
    - Each module hards to be replaced to better one.



Workflow Manager for OKBQA

- What if soft-wired by WFMS?

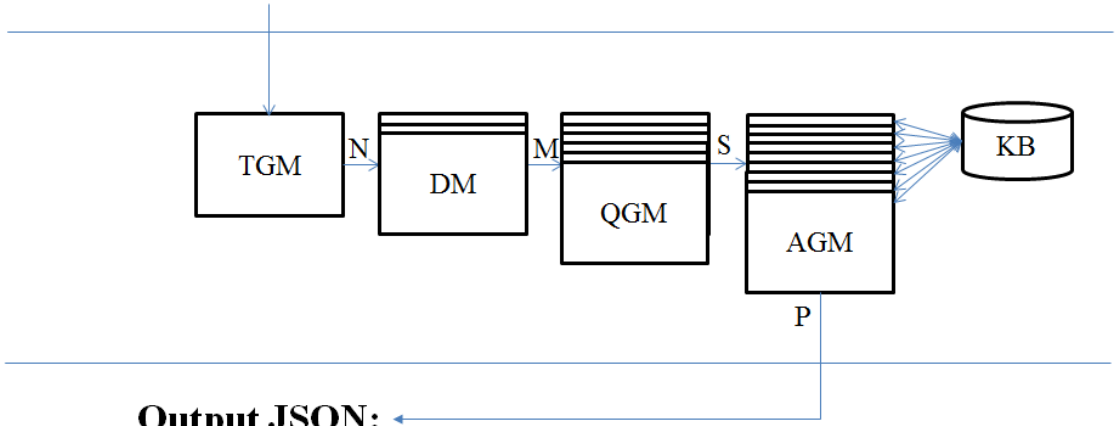


Controller: Workflow Manager for OKBQA

- A workflow management module
  for integrating OKBQA modules by soft-wiring
  - It interlinks I/O between TGM, DM, QGM, AGM and KB.

Input JSON:

{"string": "question", "language": "en"}



Output JSON:

```
{
        "log": [log(1), log(2), …, log(n)],
        "answers": [answer(1), answer(2), …, answer(n)]
}
```

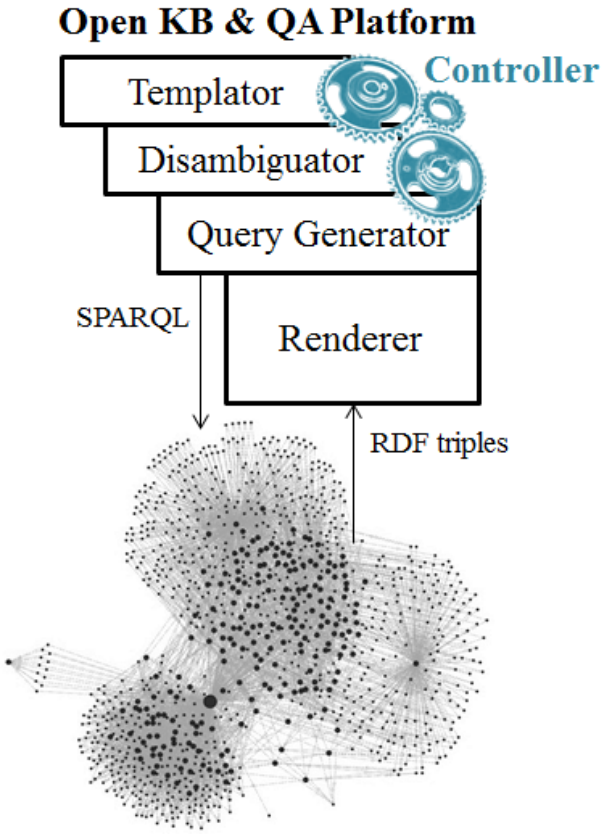
Advantages of The Dedicated WFMS

- Collaborative developments
  - **Distributed** developments
    - Each module can be developed **separately**
  - Easy **integration**
    - **I/O linking** between modules
      - –After developments done, the controller integrates modules in an united system.
  - **Logging** and **Alarming**
    - If integration fails, the controller **alarms** developers by **log-messages** to chase a **cause of fails**

What is the “key functions”?

Controller: Essential Functions

- The module for integrating OKBQA modules
  - It interlinks I/O between TGM, DM, QGM, AGM, and KB.
    - **+ RESTful API supports**
    - **+ Fault tolerance**
      - –I/O formats
    - **+ Fault alarming** (when not-tolerable)
      - –Encodings
      - –Exceptions
      - –I/O formats (hard to be corrected)
    - **+ Flexible configuration**
      - –Module addresses
      - –Graph URIs of KBs
      - –Module time-out
    - **+ Logging**
      - –I/O flows
      - –Answers for each SPARQL query
      - –Pretty-printing



Controller API & Usage

Three level Interfaces

- API 1: Command-line
- API 2: RESTful service
- GUI: Web interface

API 1: Command-line

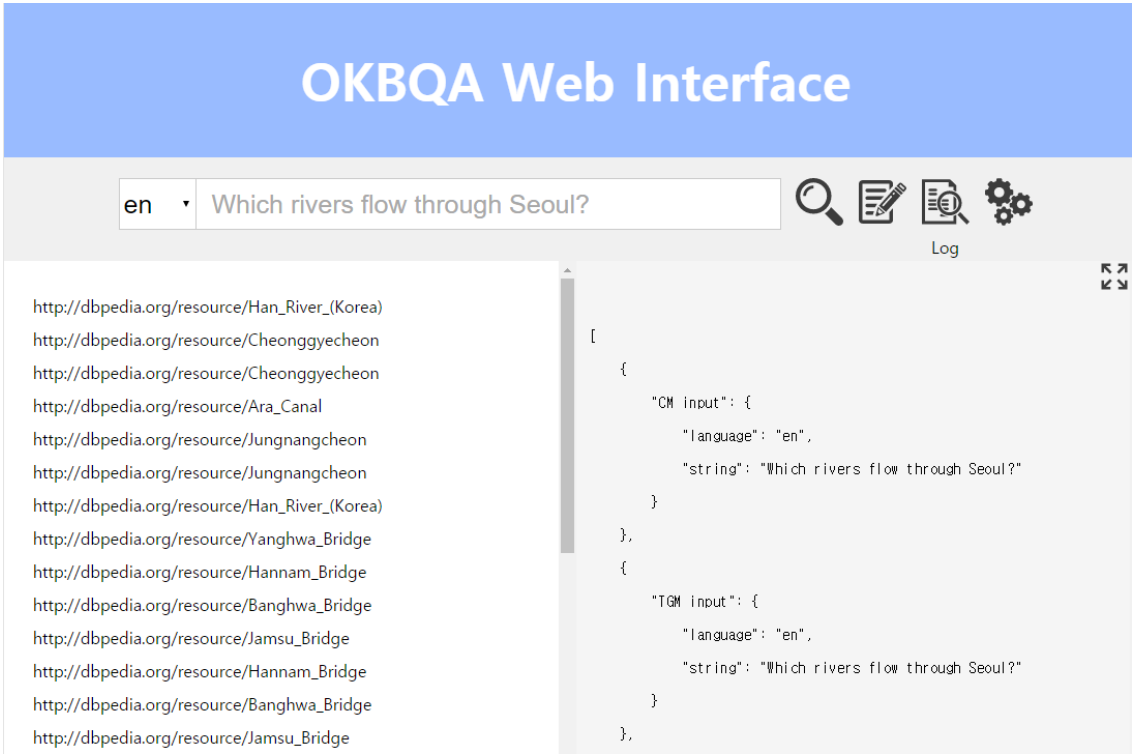
- It supports command-line API to applications.
  - Usage
    - Input JSON: {“string”: x, “language”: y, “conf”: z}
      - –x = “A natural language question”
      - –y = “ko” / “en”
      - –z = { “tgm_addresses”: [tgm(1), tgm(2), …, tgm(n)],
        ...
        “kb_addresses”: [kb(1), kb(2), …, kb(n)],
        “timeout”: t }
    - Execution: python cm_terminal.py “Input JSON”
    - Output JSON: {“log”: x, “answers”: y}
      - –x = [log(1), log(2), …, log(n)]
      - –y = [ans(1), ans(2), …, ans(n)]

API 2: RESTful service

- It provides RESTful API to the applications.
  - Built on the top of the command-line API
- Program: cm_rest.py
  - Usage
    - Input JSON: {“string”: x, “language”: y, “conf”: z}
    - Execution: python cm_rest.py
    - Output JSON: {“log”: x, “answers”: y}
      - x = [log(1), log(2), …, log(n)]
      - y = [ans(1), ans(2), …, ans(n)]

GUI: Web Interface

- It provides GUI to the developers.
  - Built on the top of the RESTful API
  - Service: http://ws.okbqa.org/~testuser02/

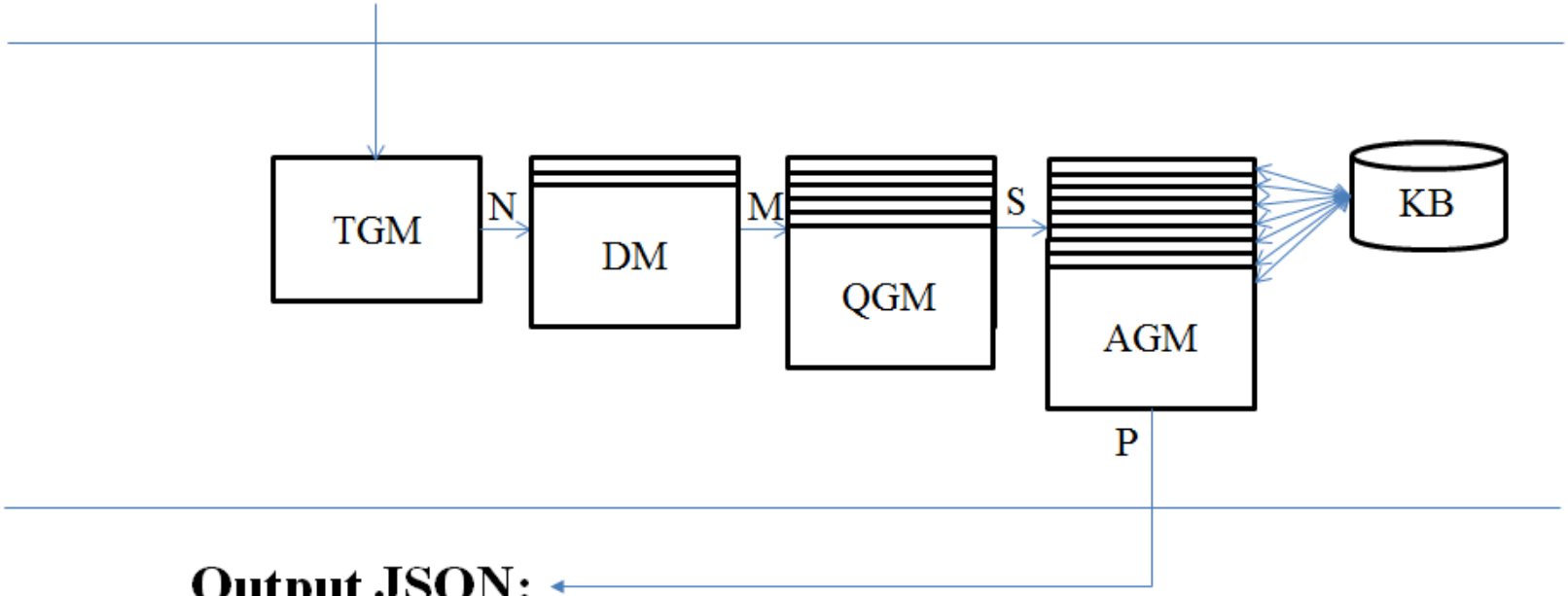


Applications with Extension

Current Architecture

Input JSON:

{“string”: “question”, “language”: “en”}



Output JSON:

{

“log”: [log(1), log(2), …, log(n)],

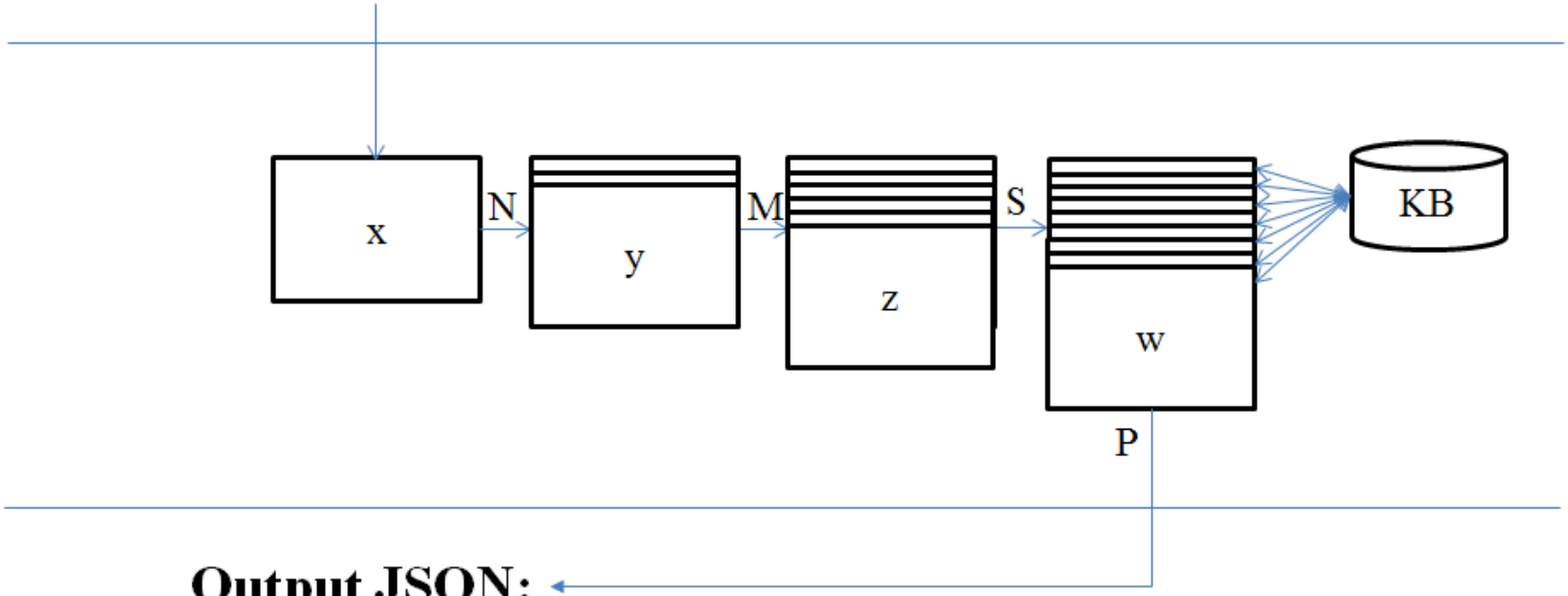
“answers”: [answer(1), answer(2), …, answer(n)]

}

Current Architecture

Input JSON:

{“string”: “question”, “language”: “en”}



Output JSON:

{

“log”: [log(1), log(2), …, log(n)],

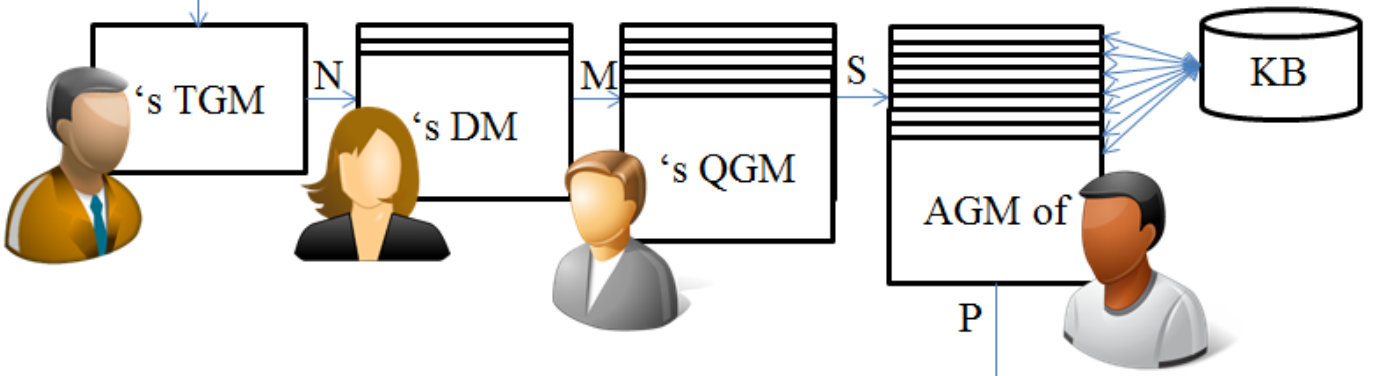
“answers”: [answer(1), answer(2), …, answer(n)]

}

Current Architecture

Input JSON:

{“string”: “question”, “language”: “en”}



Output JSON:

{

“log”: [log(1), log(2), …, log(n)],

“answers”: [answer(1), answer(2), …, answer(n)]

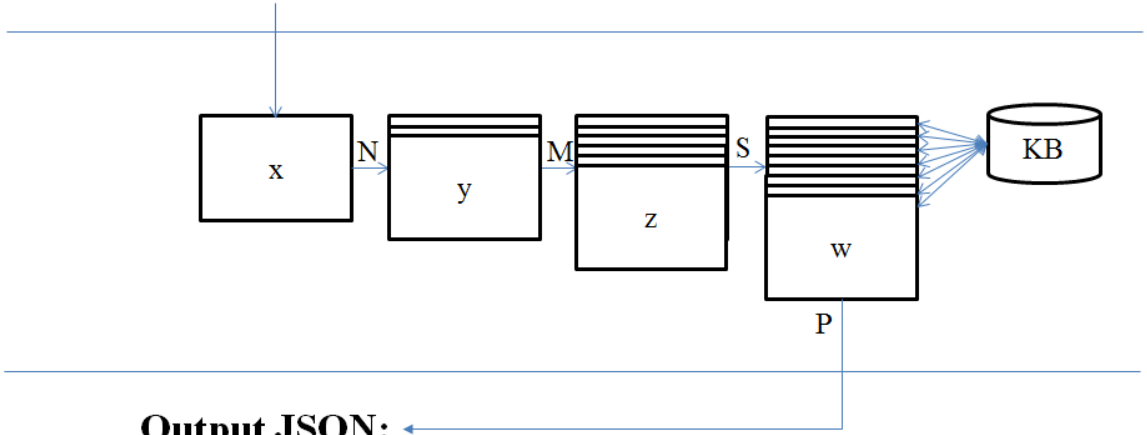
}

Current Architecture

- Limitations
  - Fixed module slots
  - Flowing I/O by one direction

Input JSON:

{"string": "question", "language": "en"}



Output JSON:

```
{
    "log": [log(1), log(2), …, log(n)],
    "answers": [answer(1), answer(2), …, answer(n)]
}
```

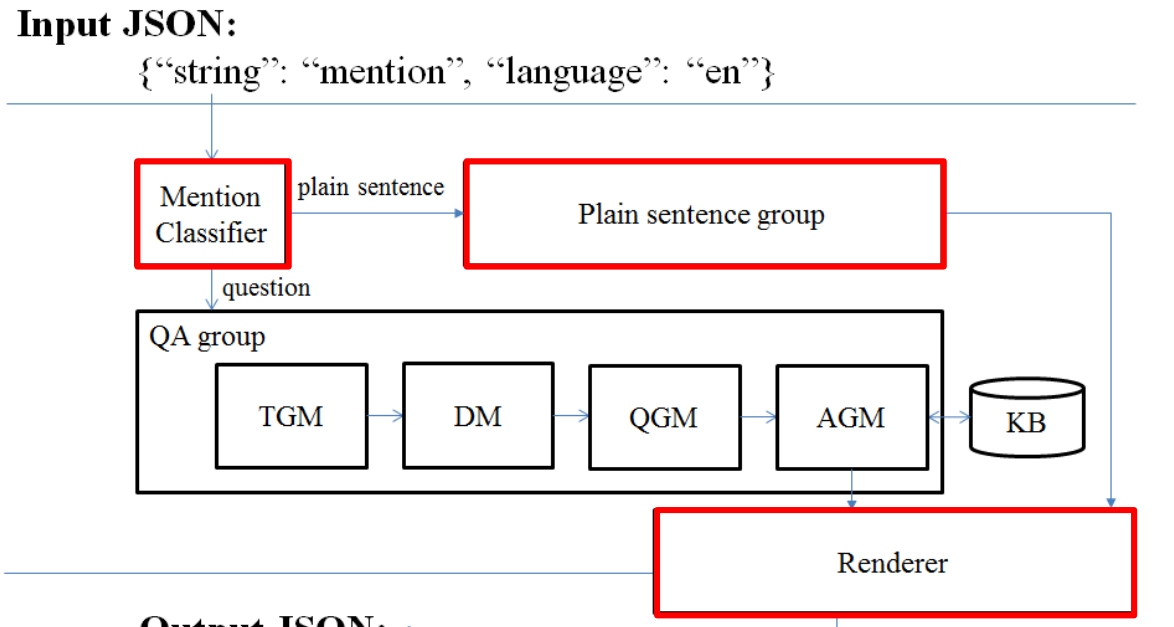
Extension

- **Key functions** as a controller
  - I/O transfer among modules
  - Fault tolerance
  - Fault alarming
  - Flexible configuration
  - Logging for debugging

- **Possible extension** for a controller
  - Module slot # configuration
    - Specifying arbitrary # of modules
  - Link configuration
    - Specifying I/O direction between modules

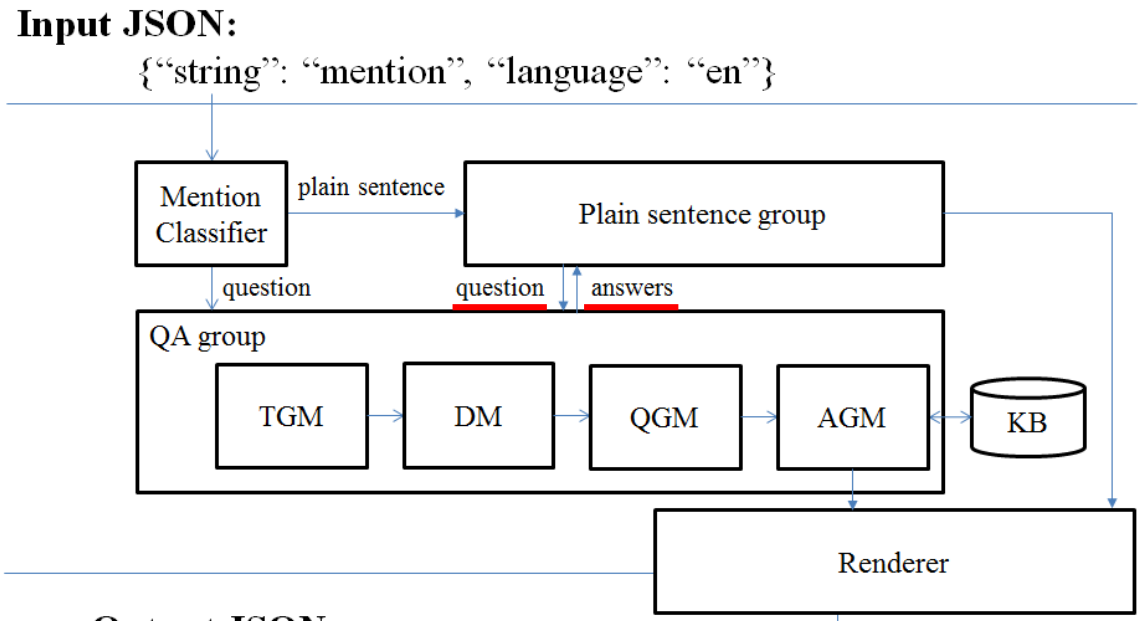
Extension 1: Module # Configuration

- For specific # of modules w.r.t. an application
  - e.g., Chatbots



Extension 2: Link Configuration

- All applications are not like waterfall.
  - e.g., Chatbots with *self-questioning*



Remind: Essential Attributes

- Collaborative developments
  - **Distributed** developments
    - Each module can be developed **separately**
  - Easy **integration**
    - After developments done,
      the controller integrates modules
      - **I/O linking** between modules
  - **Logging** and **Alarming**
    - If integration fails, the controller **alarms** developers
      by **log-messages** to chase a **cause of fails**

Hands-on: Harmonizing modules

Available Modules

- TGM
  - http://ws.okbqa.org:1555/templategeneration/templator/
- DM
  - http://ws.okbqa.org:2357/agdistis/run
- QGM
  - http://ws.okbqa.org:38401/queries
- AGM
  - http://ws.okbqa.org:7744/agm
- KB
  - http://dbpedia.org/sparql

Hands-on 1: Command-line API

Repository:
https://github.com/your-jiseong/cm_v2.0
Program:
cm_terminal.py

1. Configuring
   - Configure “m_data/conf.tsv”

1. Input JSON
   - Build your own JSON with referring to the usage page.

1. Executing
   - python cm_terminal.py “Input JSON”

Hands-on 2: RESTful API

Repository:
https://github.com/your-jiseong/cm_v2.0
Program:
cm_rest.py

1. Adjusting address setting
   - In the last line in the code “cm_rest.py”, set your machine address.
     - run(host=’IP_ADDRESS’, port=”PORT_#”)

1. Executing service
   - python cm_rest.py

1. Input JSON
   - Build your own JSON with referring to the usage page.

1. Requesting
   - curl -i -H “Content-Type: application/json” -X POST -d “Input JSON” http://IP_ADDRESS:PORT_#/cm

Hands-on 3: Web Interface

Web interface:
http://ws.okbqa.org/~testuser02/

1. Configuring
   - Click the configure button.
   - Preconfigured with the default OKBQA modules.
   - You can change setting with your own modules!

1. Input question
   - Any questions of a natural language!

1. Requesting
   - Just click the “search” button.

Wrap-up

- You **learned** about **the controller**, a.k.a WFMS.
  - Also, **experienced various interfaces** of it.
- With today's practice, you **can construct your own QA system** by the controller's supports.
- Also, you will taste **the power of the modular system architecture and design**.
  - It will make you a further **good designer**, definitely!
- Cheers! :)