GE ASKE TA1 ANSWER M11 Report Part 2: ANSWER System Documentation

Andrew Crapo, Varish Mulwad, Nurali Virani, Narendra Joshi
GE Research
October 18, 2019

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00111990006. The latest version of this report is available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/M11_ANSWER_Report_Part2.pdf.

Contents

Introduction 2
System Overview and Features 2
ANSWER Installation 4
  Installing Services 4
    Text to Triples 4
    K-CHAIN 5
    Installing Eclipse and Eclipse Plug-ins 5
Installed System Validation 6
  Services Validation 6
    Text to Triples 6
    K-CHAIN 7
  ANSWER UI and Backend Validation 8
Known Issues 8
Tutorials 8
  Code Extraction Example 8
  Text Extraction Example 8
  K-CHAIN Example 8
  Integrated Code and Text Extraction Example 9

Introduction

This report provides documentation of the ANSWER knowledge extraction system developed by GE Research as part of a DARPA ASKE TA1 project. This version accompanies the M11 code release of October 2019. It covers system design, features, installation, validation, and tutorial-style examples. The latest version of this report is available on-line at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/M11/M11_ANSWER_Report_Part2.pdf.

System Overview and Features

A functional view of the ANSWER system is shown in Figure 1.

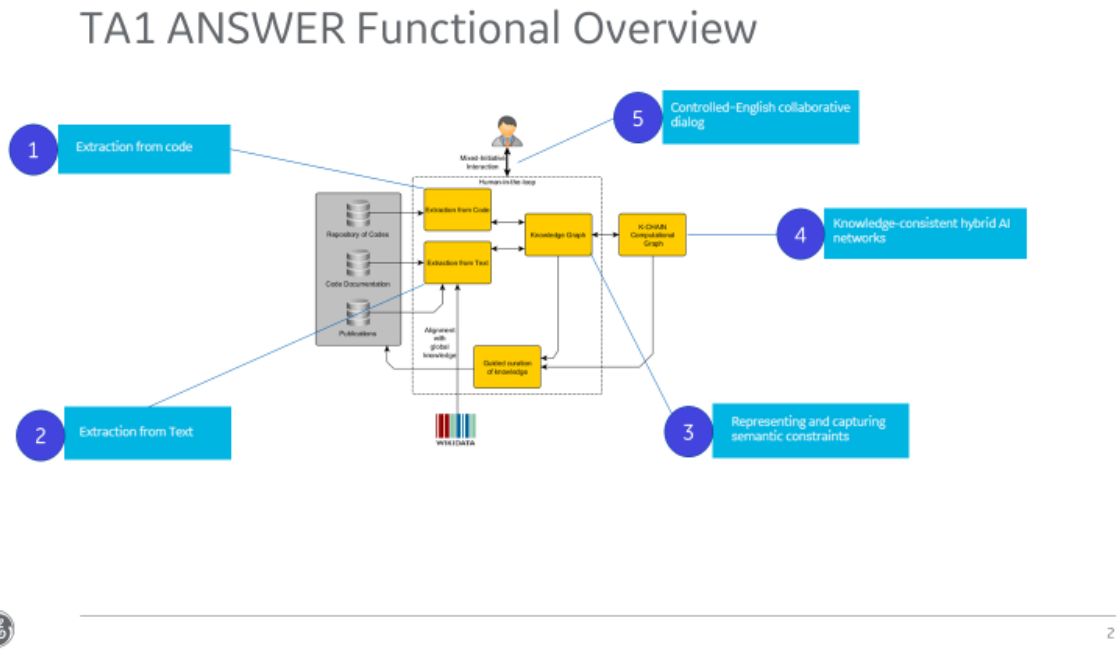

Figure 1: Functional view of ANSWER system

The primary functions of the system are:

1. Extraction of scientific knowledge and equations from Java code
2. Extraction of scientific knowledge and equations from text
3. Representation of curated knowledge and equations in a semantic knowledge graph
4. Equations and data-driven models instantiated in a computational graph for execution
5. Collaborative, mixed-initiative interaction between human and ANSWER system through a controlled-English interaction in the Dialog language.

These functions are reduced to practice through an architected ANSWER system, whose architecture is shown in Figure 2.

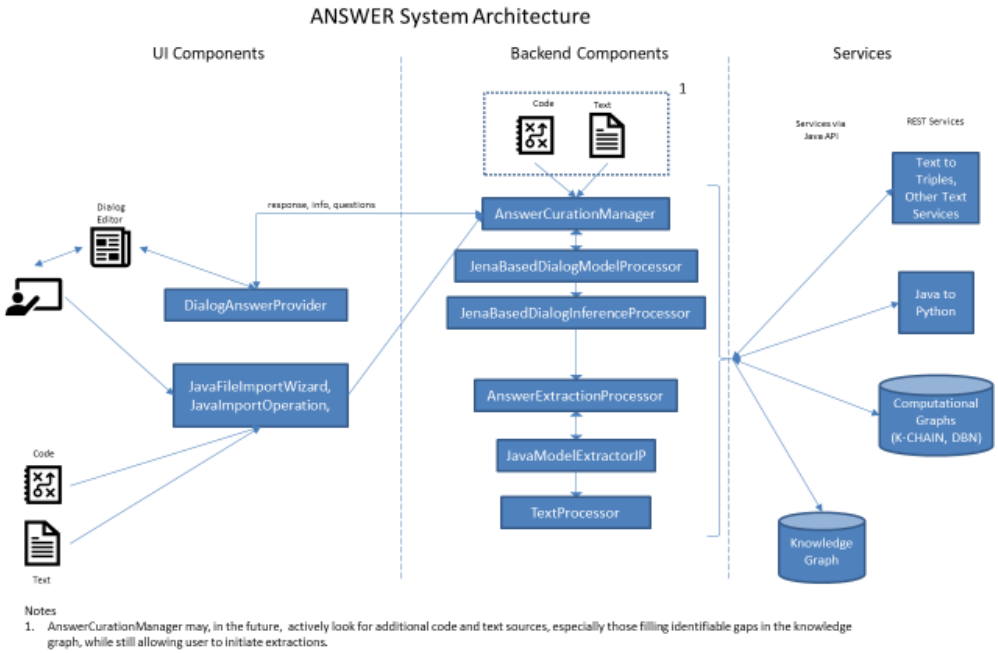

Figure 2: ANSWER system architecture

The three columns of Figure 2 show the user-interface (UI), the backend components, and the services on which the backend depends. The current user-interface is implemented in Eclipse using Xtext. The UI consists of two parts: a) the Dialog editor window in which the backend agent AnswerCurationManager and the human communicate, and b) a file import wizard that allows the user to select code and/or text files from which to extract knowledge. Note that the UI's dependency upon Eclipse is small. Xtext supports browser-based implementations and the import wizard could easily be implemented in a browser.

The backend components are managed by the AnswerCurationManager (ACM) agent, through which all interaction with the user pass via controlled English. The statements expressed in the Dialog language that are entered into the Dialog window, regardless of whether they are made by the human or by the ACM, are processed by the JenaBasedDialogModelProcessor. When questions asked by the user require inference to answer, the answers are computed by the JenaBasedDialogInferenceProcessor, using services as needed. Extraction is managed by the AnswerExtractionManager, using the Java and text extractors. The JavaModelExtractorJP uses the open source Java Parser and implements the IModelFromCodeExtractor Java Interface class. Extraction from additional languages can be added by using a parser for that language and wrapping the parsing and parse tree analysis into an implementation of IModelFromCodeExtractor.

The services upon which ANSWER depends include the following.

1. Text to triples – process a specified text and extract concepts and relationships, placing the extractions into RDF triples
2. Java to Python – take Java code and convert it into Python code

3. Computational graphs—store models, physics-based or data-driven, append them together as needed to create composite models, and perform requested computations using those models
4. Knowledge graph—store the semantic information about the domain, about all known models, the semantic meaning of their inputs and outputs, and the assumptions/constraints regarding their appropriate usage

An overview demonstration video, in two parts, is also available:

- Part 1
- Part 2

ANSWER Installation

Installing Services

Text to Triples

A copy of these instructions can be found on our GitHub repo in a README.md under ModelsFromText: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/ModelsFromText.

Pre-requisites

- Docker
- Anaconda
- Java 8

Installation

Note: the following instructions are for a Linux environment.

1. Build and run the docker container for searching over Wikidata terms. This docker container will load Wikidata terms from ModelsFromText/wikidata-for-elasticsearch into an Elasticsearch instance. To build this docker container:

```
cd wikidata-for-elasticsearch

docker build --tag=elasticsearch-aske

docker run -it -d elasticsearch-aske
```

2. Create a conda environment for text extraction services: `conda env create -f conda-env/aske-ta1-text-env.yml`
3. Run `source activate aske-ta1` to check if the environment is created.

4. Run `./getTextExtractionModels.sh` to download pre-trained model for extraction of scientific concepts and equations from text. This will create a resources folder under `ModelsFromText`.
5. Edit `getAndRunConceptMapperService.sh` and set the appropriate path to `JAVA_HOME`.
6. Run `./getAndRunConceptMapperService.sh` to download and run the concept-mapper service.
7. Run `./startServicesModelsFromText.sh` to start all the relevant text extraction services.
8. The services will be running if you are able to access its API documentation (http://localhost:4200/darpa/aske/ui/).
9. Log files for the services can be found under `logs/`.
10. Start a jupyter notebook `jupyter notebook`.
11. Example usage for text-extraction APIs and overview in each service can be found in the following notebook `api-example-code/text_to_triples_client.ipynb`

(Tip: If you are behind a firewall, make sure your proxies are set correctly)

K-CHAIN

- Dependency Installation:
  - cd into the kchain directory and verify that aske-ta1-kchain-env.yml file is present (on master branch).
  - Create K-Chain specific conda environment as follows:
    ```
    conda env create -f aske-ta1-kchain-env.yml
    ```
  - Switch conda environments to the newly created one for K-Chain:
    ```
    source activate aske-ta1-kchain-py36
    ```
- Launch service:
  - cd into the kchain directory
  - Launch kchain as follows:
    ```
    python app.py
    ```
- Test service:
  - Successful launched if able to view the service endpoints at: http://localhost:12345/darpa/aske/ui/ via a web browser

The service uses Python 3.6 and the following packages: tensorflow 1.13 (for modeling and inference), numpy (for computation), pandas (for data handling), matplotlib (for some debug plots), os (for file I/O), json (for serialization and data representation), importlib (for importing and reloading modules), connexion (for REST service setup), and requests (for interacting with REST service).

Installing Eclipse and Eclipse Plug-ins

1. Services should be installed and the URLs of each service noted
2. ANSWER has been developed and tested in Eclipse 2018-12. Download and install this or a newer version of Eclipse for Java Developers (not the DSL version) from https://www.eclipse.org/downloads/packages/.
3. Install SADL v3.2.0.20191024_DARPA_ASKE_TA1 Update from SADL release https://github.com/crapo/sadlos2/releases/tag/Answer-0.7.

   a. Download the ZIP file listed as an asset for this release.
   b. From the Eclipse Help pulldown menu, select “Install New Software…”, then click the “Add…” button on the resulting modal dialog.
   c. Click “Archive…” on the resulting dialog, browse to and Open the downloaded ZIP file.
   d. Follow the prompts to complete the installation process
4. Install DARPA ASKE TA1 ANSWER TA1-2019-10-19 from https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/releases/tag/Answer-0.7 in a manner similar to that outlined for the SADL update in the preceding step
5. From the “Window” pulldown menu, select “Preferences”
6. Click on “Dialog” in the left-hand pane
7. Add the host and port, e.g., “http://myserver.mycompany.com:1234”, for each of the services
8. Check “Use K-CHAIN” if you wish to use the K-CHAIN computational graph
9. Check “Use DBN” if you wish to use the DBN computational graph, then do steps 10 or 11, depending on where the SWI-Prolog service is to be found
10. Using DBN requires that SWI-Prolog be available as a service. If SWI-Prolog is to run on the same machine as Eclipse, download SWI-Prolog from https://www.swi-prolog.org/ and install per instructions, making sure to place the executable on the path so that it can be found when ANSWER launches the service
11. If SWI-Prolog is running on a different machine than Eclipse, do this step. Note that to do this step you must create the project that will use the SWI-Prolog reasoner (any project using DBN). This step must be done for each such project.
    a. From the Project pulldown menu, select “Properties”
    b. Expand “SADL” on the left-hand list and select “Reasoner Preferences”
    c. Select “SWI-Prolog-Reasoner” (you do not need to check it unless it is to be the default reasoner, just select it and then “Jena remains the default reasoner)
    d. Click the “Edit” button and fill in the “Prolog HTTP Service Host” and “Prolog HTTP Service Port”

Installed System Validation

Services Validation

Text to Triples

1. The text to triples services are running if you can access the API documentation at http://localhost:4200/darpa/aske/ui/ (replace localhost with a server hostname if services are deployed on a server)
2. Start a jupyter notebook `jupyter notebook`.
3. Example usage for text-extraction APIs and overview in each service can be found in the following notebook `DARPA-ASKE-TA1/ModelsFromText/api-example-code/text_to_triples_client.ipynb`. Test the services by running the cell codes in the order in which they appear. Logs can be found under `ModelsFromText/logs/`.

K-CHAIN

- We have two ways to validate that services are installed and properly comnfigured:
    1. Via Jupyter Noteook:
        - From Notebooks folder, open *Demo for KCHAIN Services.ipynb* in a jupyter notebook
        - Follow instructions to *Launch K-CHAIN Services.*
        - Run model build demonstration and verify that the output from service matches with expected output provided in the notebook.
    2. Via Swagger:
        - Once services are launched, use URL of the service (i.e. <path>:<port>) as follows: http://<url>/darpa/aske/ui/#/
        - This launches a Swagger page as follows:

        - Click on /kchain/build
        - Paste the following json as value for Parameter body:

```
{
"inputVariables": [
  {
    "name": "Mass",
    "type": "double"
  },
  {
    "name": "Acceleration",
    "type": "double"
  }
],
"outputVariables": [
  {
    "name": "Force",
    "type": "double"
  }
],
 "equationModel" : "Force = Mass * Acceleration",
 "modelName" : "Newtons2LawModel"
}
```

        - The expected Response Body is:

```
{
"metagraphLocation": "../models/Newtons2LawModel",
"modelType": "Physics",
"trainedState": 0
}
```

ANSWER UI and Backend Validation

To validate the ANSWER UI and Backend components, and the backend's interaction with the services, run the Integrated Code and Text Extraction Example described below under Tutorials.

Known Issues

1. Extraction from text accomplished through the Eclipse ANSWER file import wizard has a persistence issue that requires that Eclipse be exited and reopened before extracting from a different text file.
2. Closing and reopening a Dialog (.dialog) file may result in a NullPointerException in the Dialog document code. Relaunch Eclipse if this issue is encountered. It can be avoided until resolved by editing rather than closing the Dialog file.

Tutorials

Code Extraction Example

1. Download any Java file found on the NASA Web site and extract equations following the approach outlined in the Integrated Code and Text Extraction Example below. A set of extracted files may be found at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/JavaCodeAndTextFiles.

Text Extraction Example

1. Example usage for text-extraction APIs and overview in each service can be found in the `DARPA-ASKE-TA1` GitHub repo in the following notebook `ModelsFromText/api-example-code/text_to_triples_client.ipynb`. Test the services by running the cell codes in the order in which they appear. Logs can be found under `ModelsFromText/logs/`.

K-CHAIN Example

1. In master branch, K-CHAIN has stable build and evaluate services that can be tested and explored using Jupyter Notebook located here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/Notebooks/Demo%20for%20KCHAIN%20Services.ipynb. After cloning repository:
    a. Launch jupyter notebook from Notebooks folder and open *Demo for KCHAIN Services.ipynb* for an interactive experience, or
    b. Open *Demo for KCHAIN Services.html* to get usage documentation and get an overview of capabilities.
2. However, some advanced capabilities with respect to graph append services are not yet available on master branch and those can be tried out by checking out *development_kchain* branch. After cloning repository:
    a. Launch jupyter notebook from Notebooks folder and open *Demo for KCHAIN Services-Beta Release.ipynb* for an interactive experience, or
    b. Open *Usage Documentation for KCHAIN Services-Beta.html* to get usage documentation and get an overview of capabilities.

Integrated Code and Text Extraction Example

1. Download the Java file Mach.java and the text file Sound.txt from https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/JavaCodeAndTextFiles.
2. Store Mach.java and Sound.txt in any convenient location but NOT within the Eclipse project into which you plan to import them as you cannot import from within a project.
3. Download the M11Demo project ZIP file from https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/raw/master/StatusReports/Phase2/M11/M11Demo.zip.
4. Import the M11Demo project into the installed Eclipse instance
    a. Select "Import…" from the File pulldown menu
    b. Select "Existing Projects into Workspace" from the General folder
    c. Click on "Select archive file:", Browse to the downloaded file ZIP downloaded in step 2
    d. Select the M11Demo project and finish the import.
5. Expand the project in the Package Explorer tab and open "demo.dialog". You should be prompted to add the Xtext nature to the project—respond "yes".
6. From the File pulldown menu, select "Import…"
7. Open the "Java Model Importer" folder, select "Java Files", then "Next"
8. For the "From directory:", browse to the folder into which you expanded NASASourcesAndText in step 2
9. Check "Mach.java" and "Sound.txt"
10. For the "Into folder:" browse to the current project, M11Demo
11. Click on Finish
12. Two equations should be extracted from Sound.txt and 5 equations from Mach.java. There should be no errors.
13. At the end of the demo.dialog window content, enter "save equation_1." You should get an error marker to the left of the statement indicating that there is no specified target model.
14. After the "uri …" statement at the beginning of the file, enter statement
    a. target model "http://sadl.org/Curated.sadl".
15. The saving of equation_1 should now e successful so the line in the Dialog window becomes the following, indicating that a physics-based model has been successfully saved to the K-CHAIN computational graph
    a. `save equation_1.  CM: "Successfully built model 'equation_1', type=Physics, location=false".`
16. Enter the following statement to test the model
    a. `evaluate equation_1(286, 273.15,1.4).`
17. The answer "`CM: "[330.7102357]".`" should be the response.