QANUS

A OPEN-SOURCE QUESTION-ANSWERING FRAMEWORK

NG, Jun Ping

National University of Singapore

junping@comp.nus.edu.sg

26 January 2010

The latest version of QANUS and this documentation can always be downloaded from

http://junbin.com/qanus

Please do send in your valuable comments and feedback through the above site, or drop an email to

junping@comp.nus.edu.sg

I would like to acknowledge the guidance of Assoc. Prof. Kan Min-Yen who has provided valuable feedback and supervision, without whom QANUS might never have been completed. Thank you!

CONTENTS

Motivation and Objectives

Question-Answering (QA) systems are systems which return an actual answer in response to user queries, instead of a ranked list of documents. There has been a significant amount of research into QA systems, which culminated in the inaugural QA track of the Text Retrieval Conference (TREC) in 1999. The QA track featured annually in TREC until 2008 when it was moved to the Text Analysis Conference (2008).

Despite the huge amount of literature and work done in this area, there has been till date no effort to develop a generic question-answering framework with which researchers can use as a quick test-bed to verify the efficacy of new algorithms and methods. QANUS (Question-Answering by the National University of Singapore, pronounced *KAY-NESS*) is built specifically to meet this need.

QANUS is a open-source QA framework built with extensibility in mind. QANUS is also shipped with a reference implementation of a QA system QA-SYS which is powerful enough to act as a baseline QA system, upon which new algorithms and modifications can be quickly made. We hope that with the QANUS framework, new researchers can skip the often arduous process of building up a QA system from scratch, and instead immediately put their ideas and concepts to test.

Architecture

Overview

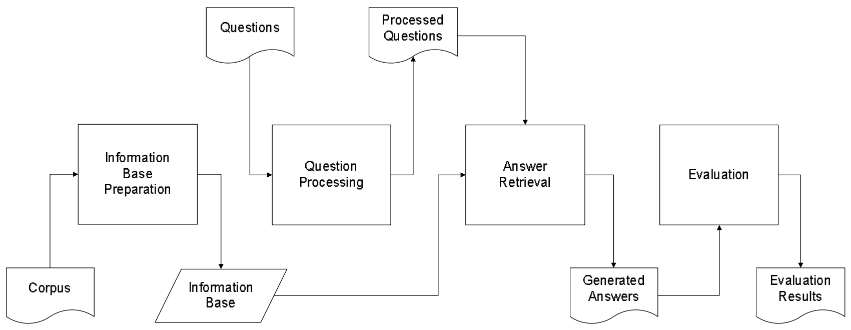

Figure 1. Overview of QANUS.

Figure 2 above shows the general architecture of the QANUS framework. QANUS adopts a pipelined approach to QA, dividing the QA task into several sub-tasks including (1) information base preparation, (2) question processing, (3) answer retrieval and (4) evaluation.

In the information base preparation (IBP) stage, an information source from which answers are to be derived can be set-up. Depending on the required needs and specifications of the QA system, the eventual information source can be a LUCENE[1] index of a corpus of documents, a full-fledged ontology, or the dynamic web. Any necessary pre-processing of the documents that will make up the eventual information source is done here. In the case where the web is used as a information source for QA, this stage may be optional if no pre-processing needs to be carried out.

The next stage is the question processing stage. Typically, questions posed to the system need to be parsed and understood before answers can be found. Necessary question processing is carried out here. Typical operations here include forming a query from the posed questions to the knowledge base, question classification to determine the expected answer type, as well as possibly parsing and part-of-speech tagging. The outputs of these various operations are stored so that they can subsequently be used by the next stage of the QA pipeline.

Subsequently, the answer retrieval stage makes use of the annotations from the question processing stage, and looks up the information source for suitable answers to the posed questions. Proper answer strings that can answer the questions are extracted in this stage. If desired, answer validation can be performed too.

With the three stages above, QANUS provides the support necessary for a fully functional QA system. The evaluation stage is introduced to complement the earlier stages and make it easy to verify the performance of the developed QA system. It is optional and may be omitted if desired. The evaluation stage cross-checks the answers computed previously by the answer retrieval stage with a set of gold-standard answers. The results of the evaluation are then output for easy review.

In this pipeline, output data from each stage is forwarded to succeeding states. The format of this data is not specified. We have deliberately avoided making any commitments in the framework to specific file formats to maximize flexibility.

[1] Open-source text search engine written in Java --- http://jakarta.apache.org/lucene

Building The Information Source

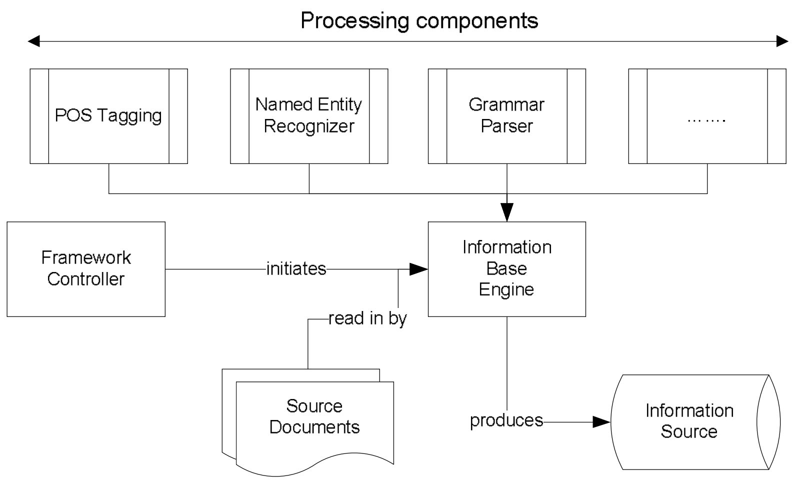

Figure 2. Preparing the Information Source.

Let us now take a closer look at each of the individual stages. To start off, we will examine the information base preparation (IBP) stage. To recap, this stage is responsible for setting up an information source which can be used in subsequent stages. Depending on your requirements, this information source can be a search engine index over a corpus like AQUAINT, or it can even be the web.

The processing components seen in Figure 2 are invoked by the `InformationBaseEngine` to act on all input from the source documents. These components provide additional information that are needed to build the information source. As an example, if you were building a search engine index of a text corpus, these processing components can include a part-of-speech tagger and a named entity recognizer. The output from the components can then be indexed too.

If however no processing is desired, this stage can be skipped too. As an example, suppose the information source is to be the web. The dynamic nature of the web means that data from the web cannot be pre-downloaded for processing. There thus will not be a need to perform this stage.

QANUS ships with some implemented processing components, and researchers are welcomed to create their own to extend the capability of QANUS. The components that come with QANUS are documented in the section *Evaluation*

The architecture of the components in the evaluation stage is shown in Figure 11.

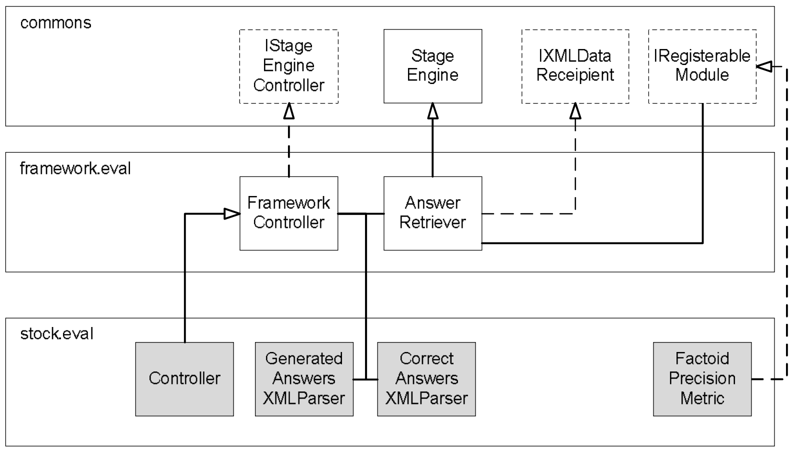

FIGURE 11. ARCHITECTURE OF EVALUATION STAGE.

The `Controller` is the entry point to the stage. It helps invoke `AnswerRetriever` which reads in a set of gold-standard answers and a set of answers generated by the earlier three stages of QA-REF. This is achieved with the help of two XML parsers `CorrectAnswersXMLParser` and `GeneratedAnswersXMLParser`.

AnswerRetriever makes use of registered `EvaluationMetric` components to gather the required statistics of the evaluation. In the case of QA-REF, it currently handles factoid questions only and so only one metric component `FactoidPrecisionMetric` is implemented. This component counts the number of correctly generated answers by QA-REF, and outputs the accuracy achieved. Accuracy in this case is defined as the ratio between the number of correctly generated results to the total number of factoid questions sent to the QA system.

Stock Text Processing Modules below.

QUESTION PROCESSING

The question processing (QP) stage is responsible for analyzing and understanding questions posed to the QA system. To accomplish this, the stage takes in provided questions, and subsequently make use of a variety of processing components to generate additional information that can help in interpreting the questions.

As an example, given a question it will be useful to know the expected answer type of the question. A processing component that can provide this information is used to provide this information.

Figure 3 illustrates how the question processing stage is structured. The keen-eyed reader will recognize that this architecture is very similar to that for the

IBP stage earlier. This uniformity is intentional to make the QANUS framework easier to understand and pick up.

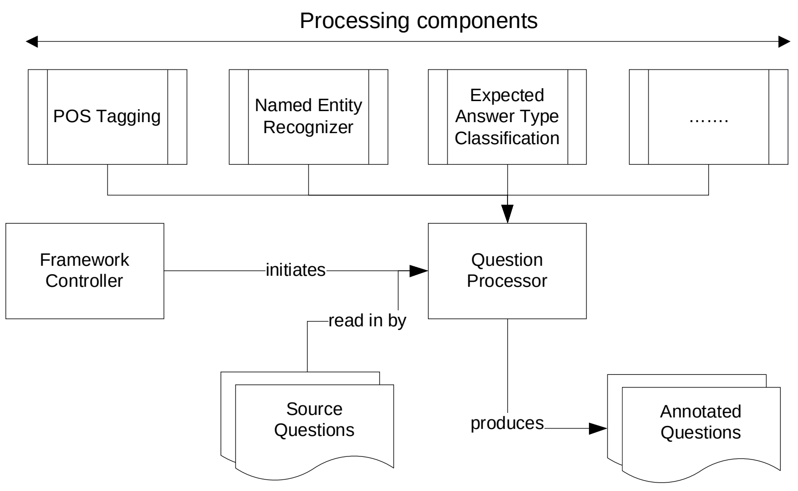

FIGURE 3. PROCESSING INPUT QUESTIONS.

Input questions in are fed to the `QuestionProcessor` which then passes the documents through various processing components as shown in the figure. The output from the various components are called *ANNOTATIONS*, and these annotations and the original questions are stored for use in subsequent stages in the QA pipeline.

ANSWERS RETRIEVAL

Making use of the information source from the IBP stage and the annotated questions from the QP stage, the answer retrieval (AR) stage attempts to find answers to the various questions.

The architecture shown in Figure 4 again exhibits a high similarity to that used in the previous two stages explained above. The `AnswerRetrievor` reads in the information source and annotated questions, and sends these information to various strategy components. Each of these components can make use of the provided information to derive answers to the posed questions. If more than one of these components is used, the `AnswerRetrievor` class chooses one answer amongst the various proposals by each component. The final answers to the questions are then output.

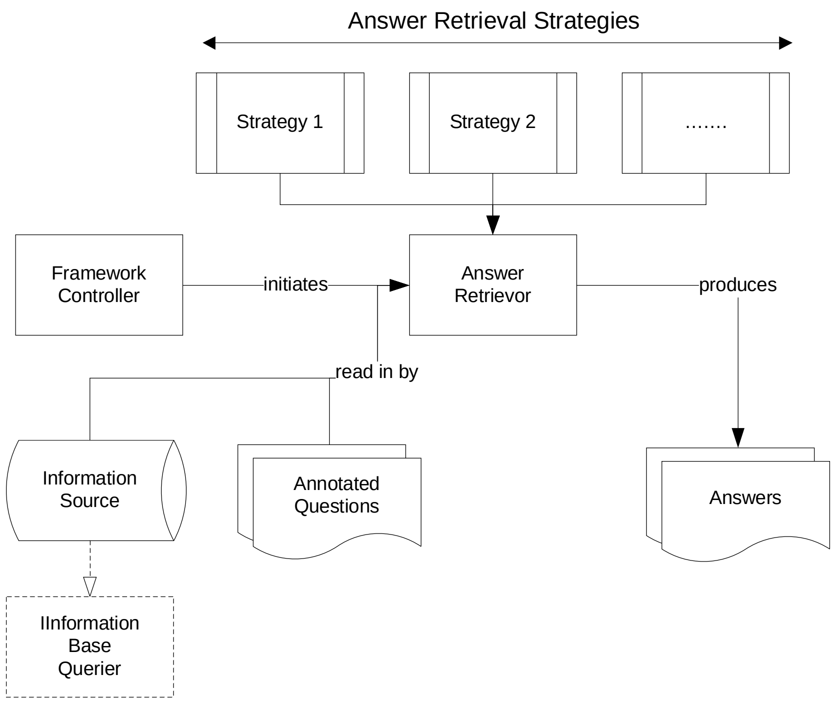

FIGURE 4. GETTING ANSWERS FROM INFORMATION SOURCE AND QUESTIONS.

There are two issues that may need to be elaborated on.

1. In instances where the information source is not built in the IBP stage, an implementation that observes the `IInformationBaseQuerier` interface needs to be provided. This interface specifies the interaction between the information source and the `AnswerRetrievor` and needs to be observed before the `AnswerRetrievor` can invoke the information source.
2. The architecture allows for multiple answer retrieval strategies to be used. However we have not implemented the logic to choose between the answers provided by different strategies.

EVALUATION

With the three stages above, QANUS already provides the support necessary for a fully functional QA system. This evaluation stage is introduced to complement the earlier stages and make it easy to verify the performance of developed QA systems. It is optional and may be omitted if desired. The stage cross-checks the answers computed previously by the answer retrieval stage with a set of gold-standard answers and outputs the results of the evaluation for easy review.

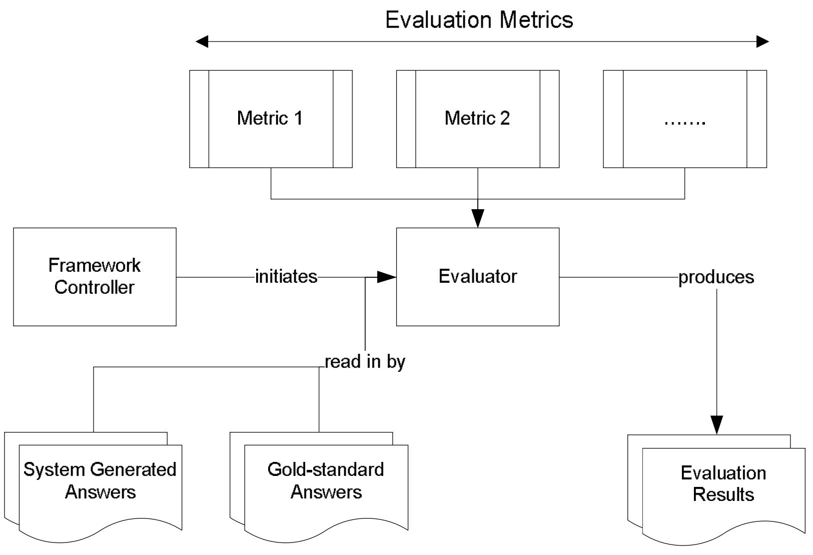

FIGURE 5. EVALUATING GENERATED ANSWERS AUTOMATICALLY.

Figure 5 shows the architecture of the evaluation stage. The answers generated by the AR stage, along with a set of gold-standard answers are fed into the `Evaluator`. Depending on the evaluation metrics to be used, the `Evaluator` invokes the metric components, and the results as determined by the metric components are output.

DIVING INTO THE IMPLEMENTATION

One of the motivations behind the development of the QANUS QA framework is to enable rapid prototyping of new QA systems. In this section, let us take a look at the actual implementation specifics of QANUS, and learn how we can easily build extensions to improve on QANUS, or make use of the provided framework to develop a new QA system.

The discussion will be structured according to the four main stages that make up QANUS – information source preparation, question processing, answer retrieval, and evaluation. To better illuminate the discussion, we will also be centering it around a reference implementation of a QA system included when QANUS is downloaded. We will refer to this QA system as QA-REF to avoid confusing it with the actual QANUS framework.

QA-REF is developed to allow us to verify the architecture and organization of the QANUS framework. It is also a useful system to study to get familiarized with the QANUS framework. It can also be re-used if you do not desire to implement every part of a QA system yourself. You can just replace specific portions of QA-REF with your own customized code and components.

Code Organization

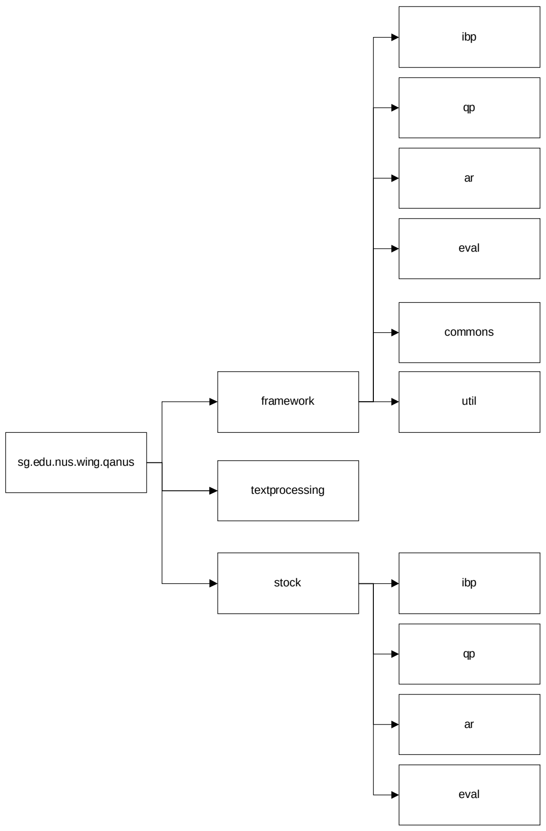

Figure 6. Organisation of QANUS source files.

The Java package structure of the source files that make up QANUS is shown in Figure 6. The names in the boxes correspond to the package name. The base package is `sg.edu.nus.wing.qanus`.

The `sg.edu.nus.wing.qanus.framework` package consists of all the components and code in the QANUS framework. These include the four stages we have explained earlier (IBP, QP, AR and evaluation), as well as two other packages `commons` and `util`. Specifications of interfaces used by the Java source files and base classes of often repeated components are implemented under the `commons` package, while the `util` package include utility functions and helping code used in the framework.

THE SG.EDU.NUS.WING.QANUS.TEXTPROCESSING IMPLEMENTATIONS OF VARIOUS TEXT PROCESSING SYSTEM. THESE INCLUDE A PART-OF-SPEECH TAGGER, WORDS, AND A NAMED ENTITY RECOGNIZER. MORE COMPONENTS CAN BE FOUND IN EVALUATION

The architecture of the components in the evaluation stage is shown in Figure 11.

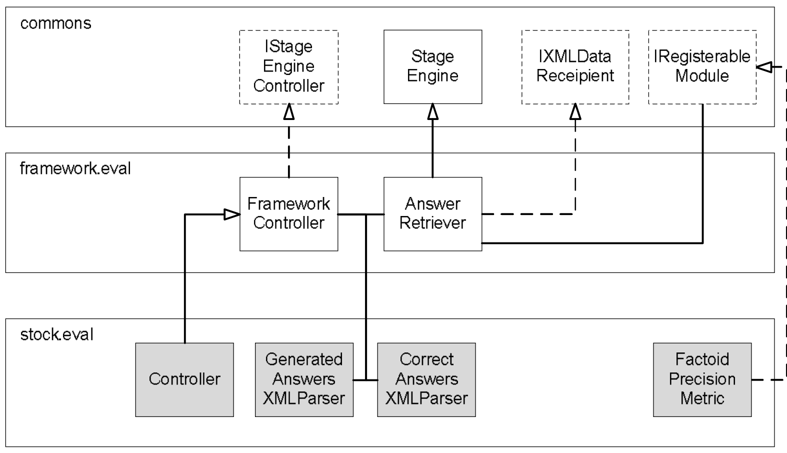

FIGURE 11. ARCHITECTURE OF EVALUATION STAGE.

The `Controller` is the entry point to the stage. It helps invoke `AnswerRetriever` which reads in a set of gold-standard answers and a set of answers generated by the earlier three stages of QA-REF. This is achieved with the help of two XML parsers `CorrectAnswersXMLParser` and `GeneratedAnswersXMLParser`.

AnswerRetriever makes use of registered `EvaluationMetric` components to gather the required statistics of the evaluation. In the case of QA-REF, it currently handles factoid questions only and so only one metric component `FactoidPrecisionMetric` is implemented. This component counts the number of correctly generated answers by QA-REF, and outputs the accuracy achieved. Accuracy in this case is defined as the ratio between the number of correctly generated results to the total number of factoid questions sent to the QA system.

Stock Text Processing Modules.

The `sg.edu.nus.wing.qanus.stock` package contains the implementation for QA-REF, our reference implementation of a QA system built on top of QANUS.

You can adopt a similar hierarchy when you set off to build a new QA system with QANUS. Keep the package hierarchy intact, removing only `sg.edu.nus.wing.qanus.stock`. You can replace it with an implementation of a QA system riding on top of the QANUS framework components.

In the sections below, the components that make up each package are explained. Some conventions that are adopted for the upcoming diagrams should be pointed out before we proceed.

1. The figures aim to educate more than detail. Thus in some instances certain components not essential to understanding QANUS or QANUS-REF may be left out for brevity and clarity.
2. The full package names are shortened to reduce clutter. In particular, the invariant portion of the package name (i.e. `sg.edu.nus.wing.qanus`) is typically dropped.
3. Following Java's convention, classes are drawn as boxes with solid outlines, while interfaces are drawn as boxes with dotted outlines. Relationships between classes or interfaces are illustrated by a solid line, inheritance is represented by a solid line with an empty arrow head pointing to the parent class, and interface observance is indicated as a dotted line with an empty arrow head pointing to the interface.

INFORMATION SOURCE PREPARATION

A detailed view of the components implemented by the QANUS framework and the QA-REF QA system is shown in Figure 7. The figure clearly highlights the various components found within the framework (`framework.ibp`), and those built on top of the framework (`stock.ibp`) to derive QA-REF.

The figure also illustrates how the QANUS framework can help reduce the effort needed to build a new QA system. Though the figure is applicable to the IBP stage, it highlights a recurring theme through all the four QANUS stages. The components within the QANUS framework typically do not need to be changed. To implement a QA system such as the example of QA-REF given here, only the shaded components are needed.

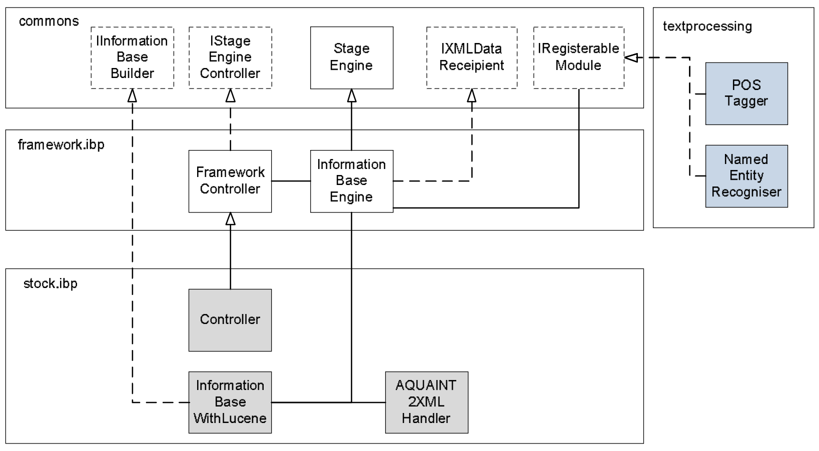

FIGURE 7. DETAILED ARCHITECTURE OF INFORMATION SOURCE PREPARATION STAGE.

Let us look at the three components implemented in `stock.ibp` which make up all the code for the IBP stage of QA-REF. The `Controller` is the entry point of the IBP stage. It takes in required arguments such as the folder in which the documents containing the information needed to build an information source are found, as well as the folder to which we want to save the prepared information source. From the development perspective, the most important task when customizing the `Controller` class is to implement the functions required by the interface `IStageEngineController`.

The functions in the interface provide the means for you to customize the text processing components used by the `InformationBaseEngine`. The processing components that are registered for use with the engine are initialized and added within these interface functions.

For example, as seen in the figure QA-REF performs part-of-speech tagging and named entity recognition on the set of input documents. The components that implement these two operations are `POSTagger` and `NamedEntityRecognizer` respectively. The part-of-speech tagger is based on the STANFORD POS TAGGER (Toutanova and Manning 2000), and the named entity recognizer is based on the STANFORD NAMED ENTITY RECOGNIZER (Finkel, Grenager and Manning 2005). `InformationBaseEngine` will invoke these registered `IRegisterableModule` components on the data that is read in from the text documents.

`InformationBaseEngine` will also output the results of the processing that is carried out into a file. This file can then be ingested and used by the next stage in the QA pipeline.

Meanwhile, `InformationBaseWithLucene` acts as the intermediary between the information source to be built and the rest of the QANUS framework. It does this by observing the `IInformationBaseBuilder` interface. For QA-REF, the information source to be used eventually is a LUCENE[2] index.

QA-REF, in the mound of other QA systems which have participated in the TREC QA track makes use of the AQUAINT corpus as an information source. The data handler class `AQUAINT2XMLHandler` is a XML parser based on the Java SAX parser and is implemented to read in and parse the XML format of the documents in the AQUAINT corpus. The Java SAX parser[3] works by way of asynchronous call-back functions. Thus `InformationBaseEngine` implements the `IXMLDataReceipient` interface which allows is to receive the call-backs when `AQUAINT2XMLHandler` is parsing through the documents from the corpus.

[2] LUCENE is an open-source Java-based text search engine. For more details please refer to http://lucene.apache.org/ .

[3] For more information on working with the SAX parser, please refer to http://java.sun.com/javase/6/docs/api/javax/xml/parsers/SAXParser.html .

So as you can see, to customize the IBP stage, regardless of the eventual information source that you hope to use, only minimal development effort is required. Much of the work that needs to be done has been abstracted by the QANUS framework.

Question Processing

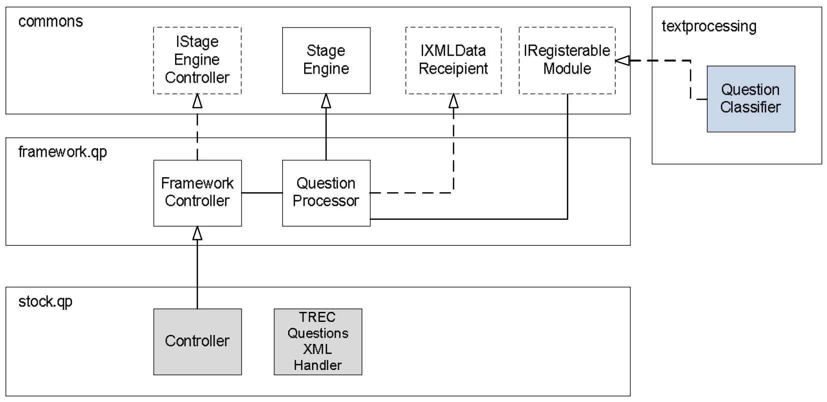

Figure 8. Detailed architecture of Question Processing stage.

Figure 8 illustrates the components that make up the QP stage. Retaining a consistent architecture as that in the IBP stage, the `Controller` is the entry point to this stage. It takes in arguments for the source folder where the question files are found, and for the destination folder to which files containing annotations to the questions are output.

The `Controller` makes use of the `QuestionProcessor` class to read in and process the input questions. The `QuestionProcessor` will send each input question to the various text processing components registered. In the case of QA-REF, only one such component --- a question classifier --- is used at the moment.

The question classifier (`QuestionClassifier`) is implemented with the Stanford Classifier (Manning and Klein 2003), using the question taxonomy explained in (Li and Roth 2002) and is responsible for finding out the expected answer type of a given question.

QA-REF works on the test questions used in the TREC 2007 QA track, which are presented in the XML format. To process these questions, `TRECQuestionXMLHandler` is implemented. This is a XML parser based on the Java SAX parser. `TRECQuestionProcessor` implements the callback function `Notify()` of the `IXMLDataReceipient` interface, where results of the annotations by the text processing components are output to the destination folder. This set of resulting annotations will be used subsequently in the succeeding AR stage.

Answer Retrieval

The AR stage again retains much of the architecture of the earlier stages as seen in Figure 9. The `Controller` is again the entry point to this stage, and is responsible for setting up the various strategy modules to use for answer extraction. It also takes in as input the annotated questions produced in the earlier QP stage.

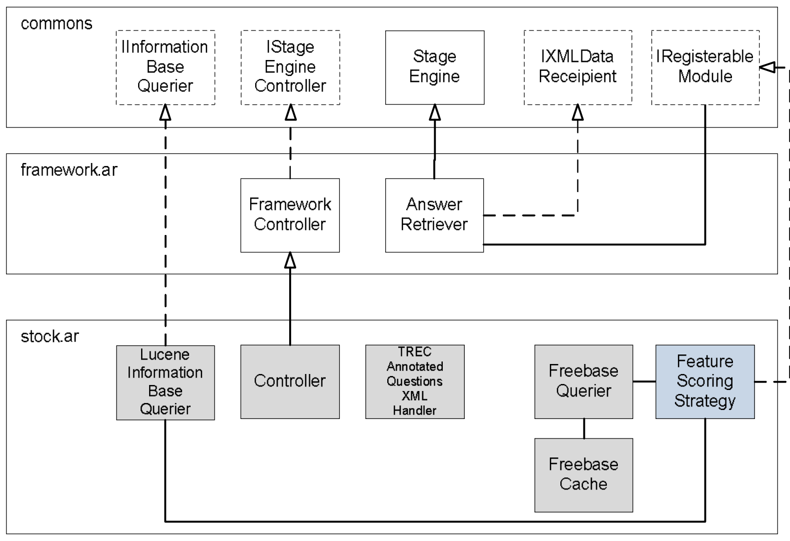

FIGURE 9. DETAILED ARCHITECTURE OF THE ANSWER RETRIEVAL STAGE.

`AnswerRetriever` makes use of the `TRECAnnotatedQuestionXMLHandler` component to read in the annotated questions expressed in XML. `AnswerRetriever` implements the `IXMLDataReceipient` to receive callbacks from this handler.

On the receipt of each question, the `AnswerRetriever` invokes the strategy modules such as the provided `FeatureScoringStrategy` component for answer retrieval and extraction. The strategy component makes use of `LuceneInformationBaseQuerier` to look up the information source built in the IBP stage. It processes the results from the query to arrive at an answer to the posed question.

Besides the LUCENE based one used here by QA-REF, other information sources can be implemented, and they can interface with the strategy component by observing the `IInformationBaseQuerier` interface.

The `FeatureScoringStrategy` component used here in QA-REF is a basic IR-based strategy for answer extraction. In this module a query string formed from the nouns and verbs of the question is sent to `LuceneInformationBaseQuerier`. `LuceneInformationBaseQuerier` makes use of the LUCENE index built earlier in the IBP stage to retrieve a set of ranked documents. The top ranked documents are then broken down into individual passages. Each of these passages is scored and ranked using a variety of heuristics such as by tabulating the occurrences of the query terms within the passages.

From the ranked passages, answer candidates are extracted depending on the expected answer type previously determined in the QP stage. The flowchart in Figure 10 illustrates how the candidates are extracted for different expected answer types. For a question seeking a person name for example, named entity recognition (NER) is used to extract people names from the ranked passages. For a question seeking a company name, NER is similarly used to find the names of organisations and businesses. For other expected answer types such as dates, hand-written regular expressions are used to aid in the extraction of answer candidates.

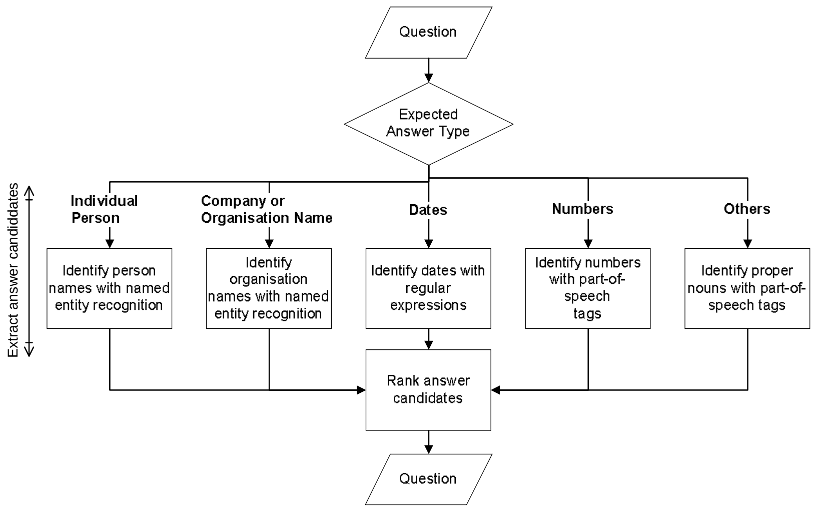
Question
Expected Answer Type
Extract answer candiddates
Individual Person
Company or Organisation Name
Dates
Numbers
Others
Identify person names with named entity recognition
Identify organisation names with named entity recognition
Identify dates with regular expressions
Identify numbers with part-of-speech tags
Identify proper nouns with part-of-speech tags
Rank answer candidates
Question

FIGURE 10. FLOW CHART ILLUSTRATING HOW ANSWER EXTRACTION IS DONE.

Finally, the answer candidates are ranked based again on a set of heuristics which include the proximity of the candidates within the ranked passages to the query terms for example. For certain answer candidates, it is also possible to perform a rudimentary form of answer validation with external knowledge sources such as Freebase[4]. For example, if the expected answer type is a country, `FreebaseQuerier` constructs a query to FREEBASE to make sure that selected answer candidates are indeed countries. To prevent QA-REF from sending queries to FREEBASE indiscriminately, a cache (`FreebaseCache`) is kept of past queries. Answer candidates which have been cached need not be sent to FREEBASE for validation repeatedly.

The highest ranked candidate is then returned as the preferred answer.

EVALUATION

[4] Open database of general information. More details at http://www.freebase.com/ .

The architecture of the components in the evaluation stage is shown in Figure 11.

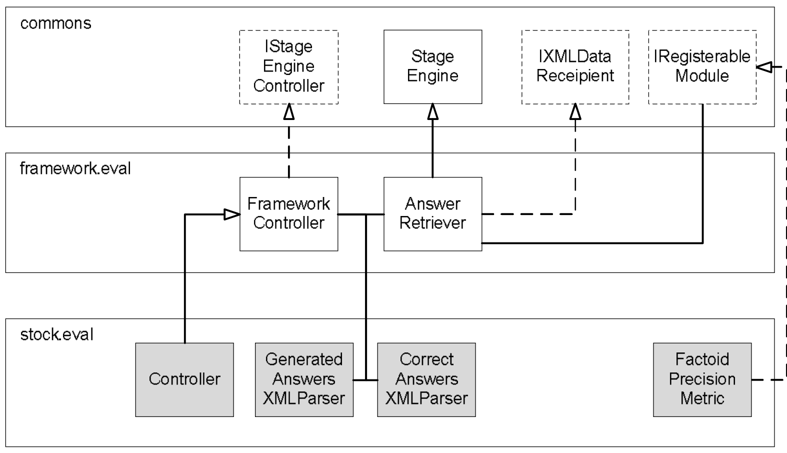

FIGURE 11. ARCHITECTURE OF EVALUATION STAGE.

The `Controller` is the entry point to the stage. It helps invoke `AnswerRetriever` which reads in a set of gold-standard answers and a set of answers generated by the earlier three stages of QA-REF. This is achieved with the help of two XML parsers `CorrectAnswersXMLParser` and `GeneratedAnswersXMLParser`.

AnswerRetriever makes use of registered `EvaluationMetric` components to gather the required statistics of the evaluation. In the case of QA-REF, it currently handles factoid questions only and so only one metric component `FactoidPrecisionMetric` is implemented. This component counts the number of correctly generated answers by QA-REF, and outputs the accuracy achieved. Accuracy in this case is defined as the ratio between the number of correctly generated results to the total number of factoid questions sent to the QA system.

STOCK TEXT PROCESSING MODULES

Out of the box, several popular stochastic text processing modules have been implemented as explained above. These modules provide the basic amount of information needed for the IR-based QANUS pipeline to process incoming questions and retrieve answers to these questions. For easy reference, the various modules that are included in QANUS are listed in Table 1.

TABLE 1. LIST OF TEXT PROCESSING MODULES SHIPPED WITH QANUS.

Text Processing Module	Implemented From
Question Classification	Stanford Classifier (Manning and Klein 2003), (Li and Roth 2002)

Part of Speech Tagging	Stanford POS Tagger (Toutanova and Manning 2000)
Named Entity Recognition	Stanford Named Entity Recognizer (Finkel, Grenager and Manning 2005)

Customizing QANUS

One of the key motivations behind QANUS is to provide a framework on which new QA systems can be quickly prototyped. To shorten the learning curve of building a new system on top of QANUS, this section highlights the key tasks and issues that will hopefully address some of the questions you may have about the framework.

Since QANUS adopts a pipe-lined design where the QA task is broken up into different stages, the discussion will be structured along the lines of the key functionalities of the system.

Information Source Preparation

For the reference implementation QA-REF, the AQUAINT-2 corpus is used, and the information source is actually a Lucene index.

Using a Different Corpus

To make use of another corpus other than the AQUAINT-2 corpus, there is a need to re-implement the XML parser `AQUAINT2XMLHandler`. If the corpus is also presented in XML, the easiest way to do this can be to just use `AQUAINT2XMLHandler` as a template. If the corpus is not XML based however, you will need to implement a new data handler to parse the input documents, and interface this with the `StageEngine`.

Currently, the architecture of the various stages are centred around the use of XML. In subsequent improvements to QANUS, we envisage that a more generic approach will be adopted to avoid committing to any one particular data exchange standard.

Building a Different Information Source

If a different type of information source is desired, you can implement an interface or wrapper for it. It is easy to integrate this new wrapper with the QANUS framework by having it observe the `IInformationBaseBuilder` and/or `IInformationBaseQuerier` interfaces.

Question Processing

Accepting Inputs of Different Formats

QA-REF makes use of the XML schema employed for the TREC-2007 test questions. If you like to make use of questions stored in a different format, you can implement a new data handler that will be able to recognize and parse the input format.

If this format is XML based also, you can trivially replace the handler that you have built with the existing one in QA-REF. If it is not, you will need to interface this manually with the `StageEngine`.

Adding Text Processing Components

To include more text processing modules to process the questions with, you can implement them using the `IRegisterable` and `ITextProcessing` interfaces and register your new modules with the `QuestionProcessor` via `Controller`. There should not be a need to make changes to the rest of the framework.

Answer Retrieval

Building Strategy Modules

The strategy module shipped with QANUS includes only basic processing techniques for answer extraction. To incorporate your own technology into the system, implement it observing the `IStrategyModule` interface, and register it with `AnswerRetriever` via the `Controller` class.

Using Multiple Strategy Modules

Currently though you can register different strategy modules with the `AnswerRetriever`, a ranking module has yet to be included to help choose between the provided answers of different strategy modules. This is because of the inherent difficulty in determining the strengths of answers from different modules.

The plan is to support this eventually through a ranking module which can be implemented separately and which will be invoked by the `AnswerRetriever` class. For now however, you will have to implement this yourself.

Evaluation

More Evaluation Metrics

QA-REF runs on the set of test questions used for the TREC 2007 QA track. If your QA system is to be run on other test sets, you may want to revise the evaluation metrics used to determine the performance of the system. To achieve this, a new class observing the `IRegisterableModule` and `IEvaluationMetric` interfaces can be created. This will allow your customized class to interface with the QANUS framework.

BIBLIOGRAPHY

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. "ncorporating Non-local Information into Information Extraction Systems by Gibbs Sampling." *roceedings of the 43nd Annual Meeting of the Association for Computational Linguistics.* 2005.

Li, Xin, and Dan Roth. "Learning Question Classifiers." *International Conference on Computational Linguistics (COLING).* 2002.

Manning, Christopher, and Dan Klein. "Optimization, Maxent Models, and Conditional Estimation without Magic." *Tutorial at HLT-NAACL and ACL.* 2003.

"SAX." http://www.saxproject.org/.

Toutanova, Kristina, and Christopher Manning. "Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger." *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora.* 2000. 63-70.