QANUS

A Generic Question-Answering Framework

NG, Jun Ping

National University of Singapore

ngjp@nus.edu.sg

30 November 2009

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

ngjp@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-NURS*) is built specifically to meet this need.

QANUS is a fully functioning QA system built with extensibility in mind. QANUS is powerful enough to act as a baseline QA system, upon which new algorithms and modifications can be quickly made. We hope that with this system, 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

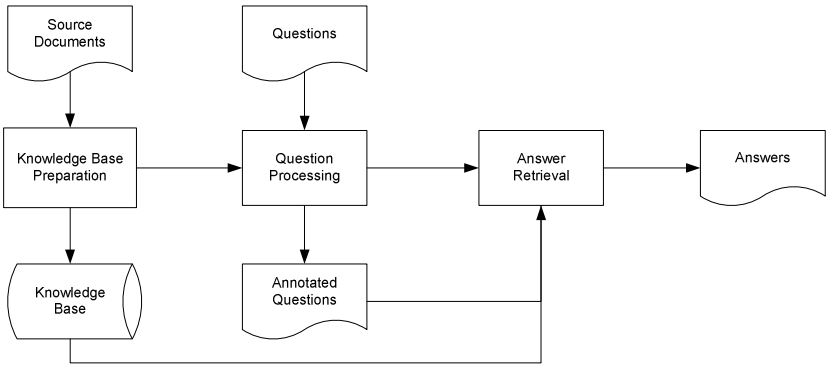

Figure 1. Overview of QANUS.

Figure 2 above shows the architecture of QANUS. QANUS makes use of a XML pipeline, where the input and output of the various components of the system are encoded as XML documents. We have deliberately avoided making any commitments in the architecture to specific file formats, or input/output channels to keep the system as general as possible. Since the output of each

component is a XML file, we believe it would be easy to customize this framework to different needs and requirements.

Before QANUS can be used to answer user questions, a knowledge base has to be built. Depending on the choice of modules used in the preparation phase, this knowledge base can be as simple as a LUCENE index of the source documents, or a full-fledged ontology.

Questions are then processed and annotated. The resulting XML document containing annotations for the questions is then sent to the answer retrieval module. The answer retrieval module makes use of the annotated information, together with the knowledge base to retrieve suitable answers.

These 3 key pipeline stages of QANUS – knowledge base preparation, question processing, and answer retrieval will be elaborated on in the following sections.

BUILDING THE KNOWLEDGE BASE

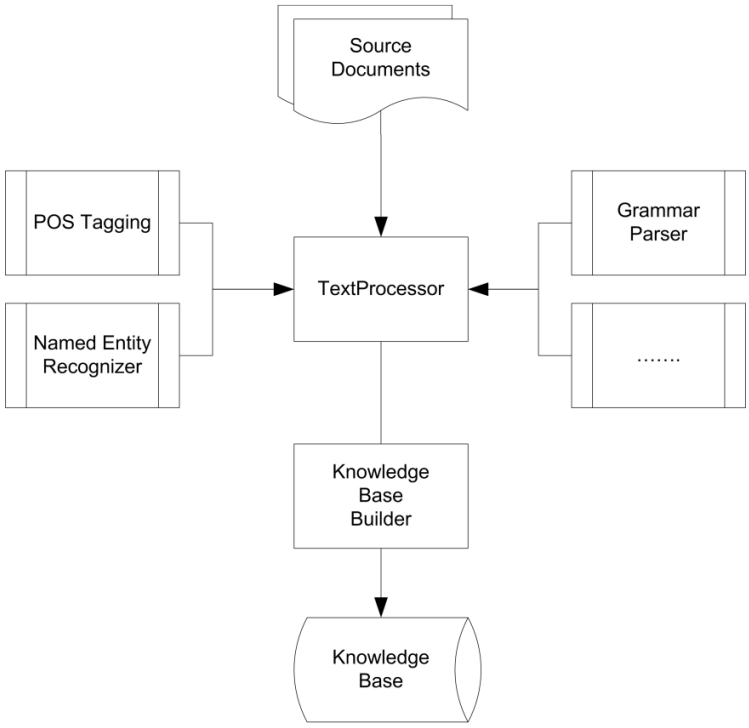

FIGURE 2. PREPARING THE KNOWLEDGE BASE.

As the first step to readying QANUS, we need to build a knowledge base out of source documents. Typically these can be a corpus like the Wall-Street-Journal (WSJ) corpus, or a custom Wikipedia corpus. The only requirement is for the documents to be encoded in XML.

The documents are passed through the `TextProcessor` module which can contain many text processing modules, ranging from part-of-speech taggers to

grammar parsers. QANUS ships with a default set of modules, and researchers are welcomed to create their own to extend the capability of QANUS. The modules that come with QANUS are documented in the section *Stock Text* Processing Modules below.

Modules installed into the `TextProcessor` are activated one at a time as the source documents are processed. Each module is invoked through a call-back function registered with `TextProcessor` and are expected to produce another XML file containing the original input from the source documents, as well as any additional mark-ups produced by the module.

Consider the following XML fragment :

```
<text>The dog ran away.</text>
```

A part-of-speech (POS) tagging module after ingesting it will produce in its XML output file a fragment such as the one shown here :

```
<text-pos>
  The/DT dog/NN ran/VBD away/IN ./.
</text-pos>
```

Subsequent modules can choose to make use of or ignore the additional information produced by earlier modules. This also thus suggests that there will be an order imposed on which modules should run ahead of others. This choice is left open for flexibility.

Eventually after all the modules have gone through the source document, the resulting XML file is sent to the Knowledge Base Builder (or KBB for short). The default implementation of the KBB shipped with QANUS understands the markup introduced by stock modules incorporated into QANUS. Researchers who add additional modules, or tweak the markup of existing modules will also have to customize the KBB to make full use of the information captured by the text modules.

How this resulting knowledge base (KB) is implemented is not specified as part of QANUS, because depending on the inference and reasoning mechanisms planned for, the implementation of the KB can vary widely. While the choice of implementing the KB is left open, note however that the `AnswerRetrieval` module needs to be aware of the KB implementation.

Questions Processing

The question processing stage is responsible for taking in questions provided in a XML document, and subsequently running them through a variety of text

processing modules. The architecture of this part of QANUS is similar to that employed in the knowledge base building stage.

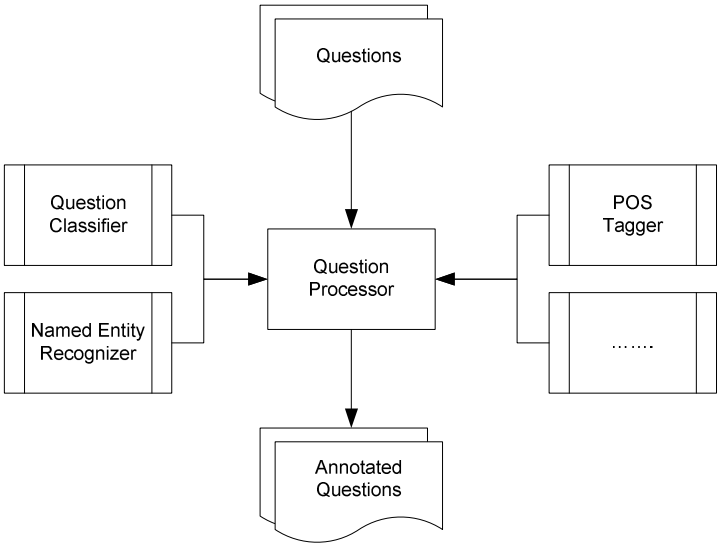

FIGURE 3. PROCESSING INPUT QUESTIONS.

Figure 3 illustrates how the question processing stage is structured. Input questions in XML documents are fed to the `QuestionProcessor` which then passes the documents through various text processing modules, including a question classifier, and named entity recognizer. The architecture is intentionally replicated from that for knowledge base building for purposes of uniformity and ease of understanding.

The various text processing modules as before add on annotations and the final XML output consists of the original question, along with the annotations of the various modules.

ANSWERS RETRIEVAL

With the knowledge base and annotated questions from the earlier two stages, the answer retrieval stage passes these information to different answer retrieval strategies. The answers returned by these strategies are then ranked and validated to get the best answer. The identified best answers are then output into a XML file.

This architecture shown in Figure 4 again exhibits a high similarity to the previous two shown above, and makes it easy to combine several techniques for answer extraction. It is straight forward to incorporate additional functionalities into the answer retrieval component, such as a results validation module for example which makes use of the World Wide Web (WWW) to verify selected answers.

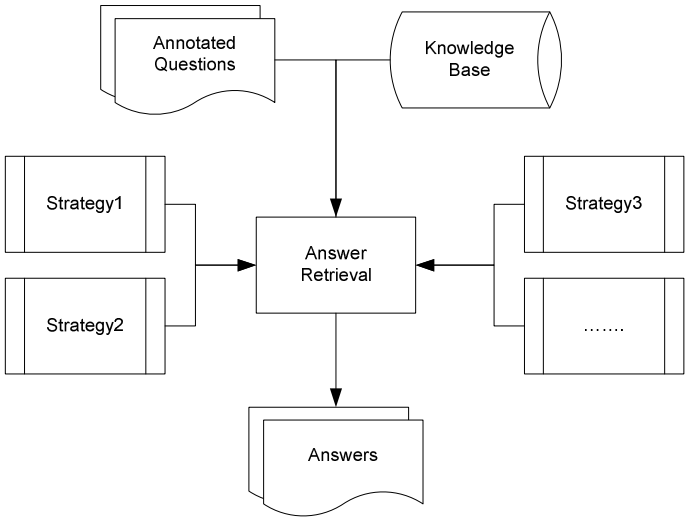

FIGURE 4. GETTING ANSWERS FROM KNOWLEDGE BASE AND QUESTIONS.

DIVING INTO THE IMPLEMENTATION

One of the motivations behind the development of QANUS 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 again along the three main modules that make up QANUS – knowledge base building, question processing and answer retrieval.

KNOWLEDGE BASE BUILDING

A detailed view of the components involved in building the knowledge base used by QANUS is shown above. They are all implemented in the package `sg.edu.nus.wing.qanus.knp`.

The `Controller` is the entry point of the knowledge base building (KBB) system. It takes in required arguments such as the folder in which the XML source documents are found, as well as the folder to which we want to save our built knowledge base.

The controller invokes the `TextProcessor` to read in and parse the XML documents. The `TextProcessor` in turn activates each registered `TextProcessingModule` to annotate the read XML. The framework is held in place by a set of interfaces like `IXMLParser` and `IXMLDataReceipient`, as well as abstract base classes like `TextProcessor`. QANUS is shipped to work on the AQUAINT corpus, and thus we have provided classes

AQUAINTXMLHandler and AQUAINTTextProcessor. To customize QANUS to work with other corpora, only these 2 files need to be updated or replaced. The rest of the framework need not be changed.

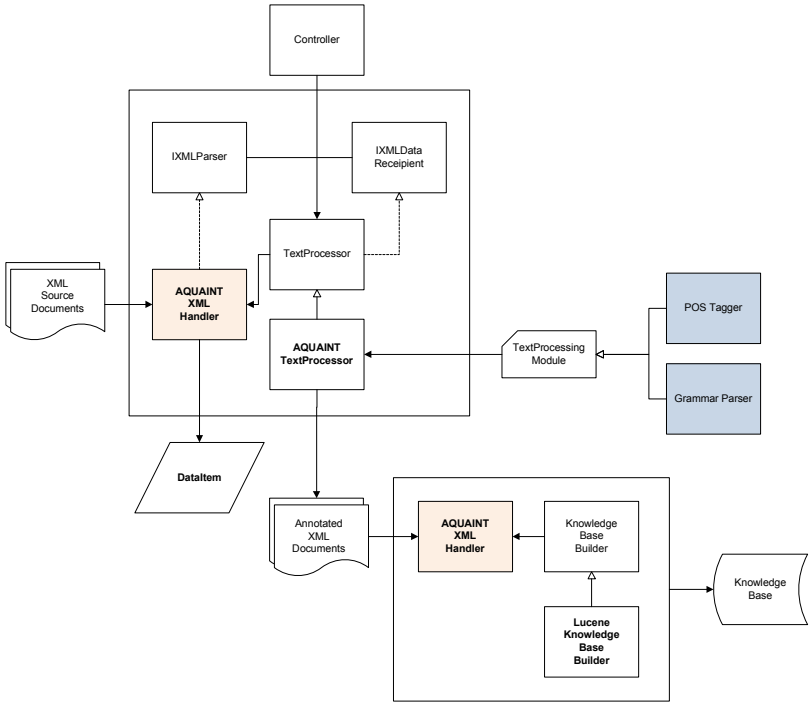

FIGURE 5. DETAILED ARCHITECTURE OF KNOWLEDGE-BASE-BUILDING STAGE.

AQUAINTXMLHandler is a XML parser based on the Java SAX parser. When working with a different corpus, a new implementation is needed to observe the schema of the corpus. AQUAINTTextProcessor implements the callback function Notify() of the IXMLDataReceipient interface, and can be modified according to the structure of the corpus to be used. Parsed XML instances from the IXMLParser are sent to the IXMLDataReceipient via this function. The AQUAINTXMLHandler boxes in the diagram are coloured to denote that they are actually the same class. The boxes are duplicated to ensure clarity.

DataItem is an abstraction of a XML instance, and can be modified accordingly to suit the structure of the XML corpus. It makes not prior commitments to the schema of the corpus, and its generality means that it can be used throughout the entire XML pipeline as an internal representation for data parsed from the XML files.

The TextProcessor will output the XML that is processed and annotated (by the TextProcessingModules) as another set of XML files. These files will then be ingested by the KnowledgeBaseBuilder. Since the annotated XML

files are similar in structure to the original XML source documents, QANUS makes use of the `AQUAINTXMLHandler` again to parse the annotated XML documents for the `KnowledgeBaseBuilder`. In general however, this could also be a totally different XML parser implementation depending on how the annotated XML files are stored.

Similarly to `TextProcessor`, `KnowledgeBaseBuilder` actually implements `IXMLDataReceipient` also (this is not shown in the diagram for brevity). Parsed XML instances from the annotated XML files are passed to `KnowledgeBaseBuilder` and `LuceneKnowledgeBaseBuilder` for processing. `LuceneKnowledgeBaseBuilder` is a derived class which implements the `Notify()` function of the `IXMLDataReceipient` interface.

For easy reading, note that the classes that need to be modified when using a different corpus than that QANUS is shipped to handle, or when defining a custom knowledge base, have their names **bolded**. Also, the `TextProcessingModules` can be implemented as required.

Through this framework, we hope to simplify as much as possible the work needed when building a knowledge base for QA. The core components of the framework need not be modified and only modules interfacing with your preferred corpora or knowledge base format need be changed. In fact we have intentionally kept to this layout for the rest of the QANUS system too to simplify the learning curve needed to understand the system.

QUESTION PROCESSING

Figure 6 illustrates the classes that make up the question processing stage. Retaining a consistent architecture with that in the knowledge base building stage, the `Controller` is the entry point to this stage. It takes in arguments for the source folder where the XML question files are found, and for the destination folder to which XML 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 modules including a question classifier, named entity recognizer, and part-of-speech tagger.

The question classifier is implemented with the STANFORD CLASSIFIER (Manning and Klein 2003), using the question taxonomy explained in (Li and Roth 2002). The named entity recognizer is based on the STANFORD NAMED ENTITY RECOGNIZER (Finkel, Grenager and Manning 2005), and the part-of-speech tagger is based on the STANFORD POS TAGGER (Toutanova and Manning 2000).

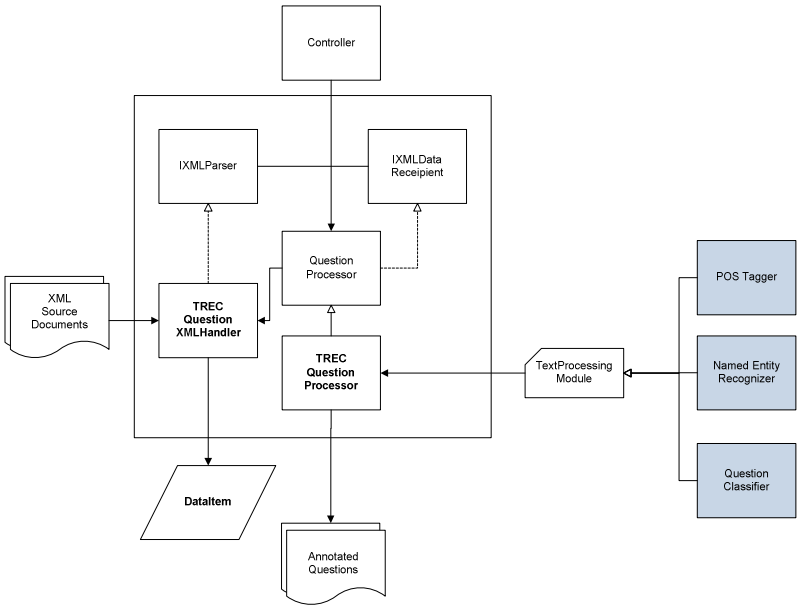

FIGURE 6. DETAILED ARCHITECTURE OF QUESTION PROCESSING STAGE.

`TRECQuestionXMLHandler` is a XML parser based on the Java SAX parser. To make use of questions files denoted with a different XML schema from that used in TREC-2007, a new implementation is needed to adapt to the different schema. `TRECQuestionProcessor` implements the callback function `Notify()` of the `IXMLDataReceipient` interface, where results of the annotations by the different text processing modules are output to the destination folder. This set of resulting annotations will be used subsequently in the answer retrieval stage.

`DataItem` as above is an abstraction of a XML instance. Since it is general enough, we also make use of this class to build an internal representation of the questions parsed from the XML files.

For easy reading, note that the classes that need to be modified when using a set of questions in a different XML schema than that in TREC-2007 have their names **bolded**.

CUSTOMIZING ANSWER RETRIEVAL

The answer retrieval stage retains much of the architecture of the earlier stages as seen in Figure 7. However there are a few differences, notably because this stage takes in the input of the earlier two stages, and not from external sources.

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, as well as the locations to locate the source XML files containing the questions we want to work on.

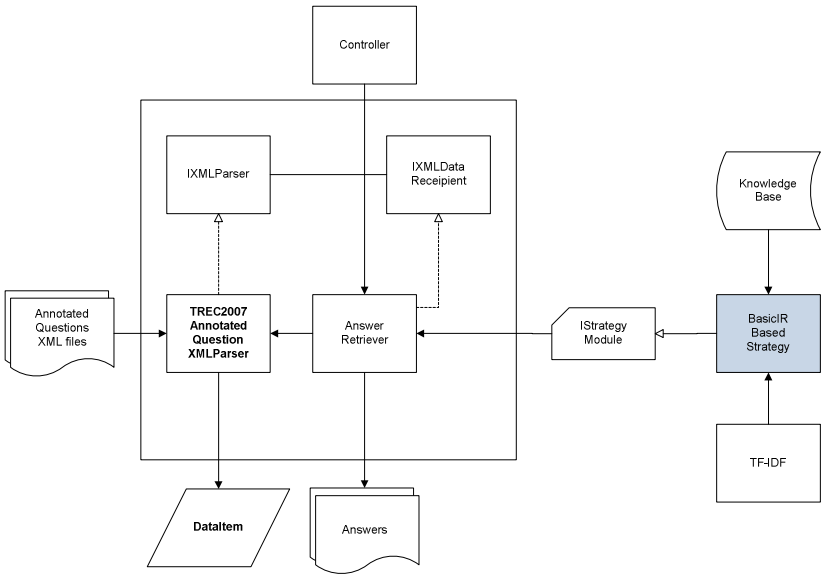

FIGURE 7. DETAILED ARCHITECTURE OF THE ANSWER RETRIEVAL STAGE.

The `AnswerRetriever` module makes use of the `TREC2007AnnotatedQuestionXMLParser` module to read in questions expressed in the XML files. The parser as before implements the `IXMLParser`, and the callback function `Notify()` required by the `IXMLDataReceipient` interface is used to send parsed questions to the `AnswerRetriever`.

On the receipt of each question, the `AnswerRetriever` invokes the strategy modules such as the provided `BasicIRBasedStrategy` module for answer retrieval and extraction. The strategy modules implement the actual functionalities of looking up the database built in the knowledge base building stage, and processing the information from it to arrive at an answer to the question at hand.

The `BasicIRBasedStrategy` module shipped with QANUS 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 the LUCENE search engine built earlier. The top 100 documents are retrieved, and a TF-IDF score is computed for every sentence within these documents. The score is used to rank the individual sentences, and the top ranked sentence is selected for actual answer extraction. Using previously annotated information from the question processing stage, such as the expected answer type of the question, as well as named entity information and part-of-speech tags, a basic rule-based heuristic is used to identify an answer string. The extracted answer strings are then output in XML format.

The `TREC2007AnnotatedQuestionXMLParser` module is **bolded** to denote that there may be a need to make changes to the module. This is especially so if there is a need to process annotated questions in a different XML format

than the default used by the question processing stage. The strategy modules need also be implemented according to the desired answer extraction strategy.

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. It is straightforward to customize QANUS by varying the type of text processing modules that are used. For easy reference, the various modules that are included in QANUS are listed the Table 1.

Table 1. List of text processing modules shipped with QANUS.

Pipeline Stage	Text Processing Module	Implemented From
Knowledge Base Building	NA	NA
Question Processing	Question Classification	Stanford Classifier
	Part of Speech Tagging	Stanford POS Tagger
	Named Entity Recognition	Stanford Named Entity Recognizer
Answer Retrieval	Part of Speech Tagging	Stanford POS Tagger

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 discusses the work required to customize QANUS for your specific needs.

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.

Knowledge Base Building

In QANUS, the AQUAINT-2 corpus is used, and the knowledge base 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 `AQUAINTXMLHandler`. All the XML

parsers in QANUS make use of the SAX API. It is chosen because of its ability to scale to large input documents, which is common for the purpose of QA. For additional information on programming with the SAX API, you may want to refer to (SAX n.d.)

Building a Different Knowledge Base

Depending on the corpus that is used, and the text processing modules that need to be run, the same or different XML parser may be used as the starting point to build the knowledge base from.

The `TextProcessor` would have output the text from the input corpus together with the annotations of the various text processing modules in a XML file.

To build a knowledge base, build a new sub-class of `KnowledgeBaseBuilder` and replace the `LuceneKnowledgeBaseBuilder` with your new module.

Question Processing

Using a Different XML Format

QANUS 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 XML format, implement a new SAX-based XML handler, and replace the `TRECQuestionXMLHandler` class with your new handler.

Adding Text Processing Modules

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

Output from these text processing modules is stored in XML files which can be used directly by the answer retrieval stage.

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 with the `IStrategyModule` interface, and register it with the `AnswerRetrieval`.

If you have not made changes to the output XML format of the question processing stage, there is no need for further changes. Retrieved answers based on your new strategy module will be output in XML to the specified destination folder when you invoked the answer retrieval stage.

Using Multiple Strategy Modules

Currently though you can register different strategy modules with the `AnswerRetrieval`, 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 `AnswerRetrieval` class. A basic ranker that can ship with QANUS may be a validation module which makes use of the World Wide Web (WWW) to verify the answers provided by the different strategy modules.

Evaluation

There is an evaluation module included with QANUS to allow us to quickly verify the correctness of generated answers.

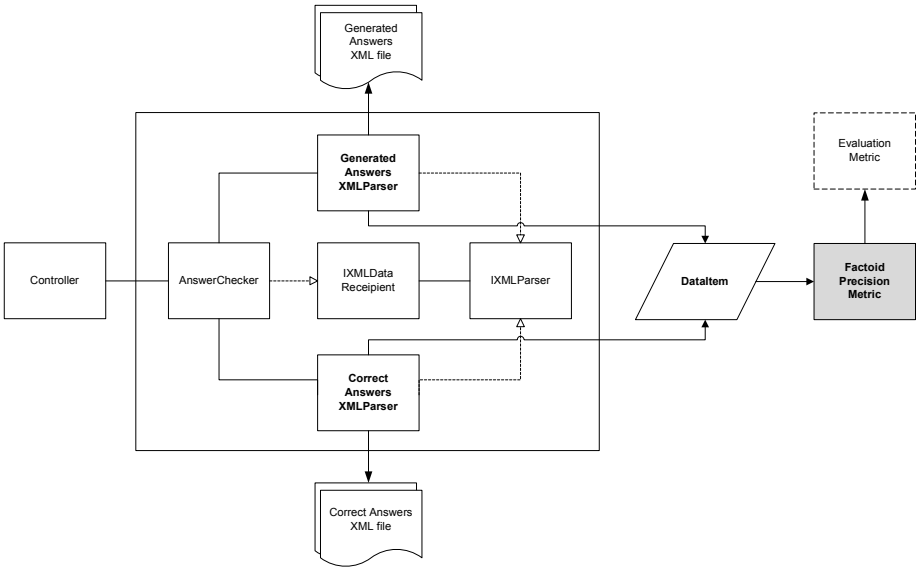

Figure 8. Evaluation module.

Figure 8 shows the structure of the module. It is architecture similarly to the different stages explained earlier. Classes with their names boldened can be changed to customize the evaluation module.

The `Controller` is the entry point for the evaluation module. It invokes the `AnswerChecker` which makes use of two different XML parsers for the two different XML input files. These input files contain the answers generated by the Answer Retrieval stage, as well as the *gold-standard* answers to evaluate against.

Currently QANUS includes a module `FactoidPrecisionMetric` which helps to tabulate the correctness of factoid questions. In particular, the XML parser `CorrectAnswerXMLParser` accepts as input a XML file containing the answers to the factoid questions in the TREC-2007 QA track. If a different answer file is used, the XML parser has to be updated accordingly.

Different scoring metrics can be measured by introducing additional `EvaluationMetric` classes besides using the provided `FactoidPrecisionMetric`.

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.