International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

F.746.11

(08/2020)

SERIES F: NON-TELEPHONE TELECOMMUNICATION SERVICES

Multimedia services

Interfaces for intelligent question answering system

Recommendation ITU-T F.746.11

ITU-T F-SERIES RECOMMENDATIONS

NON-TELEPHONE TELECOMMUNICATION SERVICES

TELEGRAPH SERVICE	
Operating methods for the international public telegram service	F.1–F.19
The gentex network	F.20–F.29
Message switching	F.30–F.39
The international telemessage service	F.40–F.58
The international telex service	F.59–F.89
Statistics and publications on international telegraph services	F.90–F.99
Scheduled and leased communication services	F.100–F.104
Phototelegraph service	F.105–F.109
MOBILE SERVICE	
Mobile services and multidestination satellite services	F.110–F.159
TELEMATIC SERVICES	
Public facsimile service	F.160–F.199
Teletex service	F.200–F.299
Videotex service	F.300–F.349
General provisions for telematic services	F.350–F.399
MESSAGE HANDLING SERVICES	F.400–F.499
DIRECTORY SERVICES	F.500–F.549
DOCUMENT COMMUNICATION	
Document communication	F.550–F.579
Programming communication interfaces	F.580–F.599
DATA TRANSMISSION SERVICES	F.600–F.699
MULTIMEDIA SERVICES	**F.700–F.799**
ISDN SERVICES	F.800–F.849
UNIVERSAL PERSONAL TELECOMMUNICATION	F.850–F.899
ACCESSIBILITY AND HUMAN FACTORS	F.900–F.999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T F.746.11

Interfaces for intelligent question answering system

Summary

Recommendation ITU-T F.746.11 describes interfaces for the intelligent question answering service framework (Recommendation ITU-T F.746.3). This Recommendation also defines the interfaces among functional modules to support the intelligent question answering service, which provides advanced functions to generate answers for the user's question in a natural language. The scope of this Recommendation is focused on describing interfaces and functional features for natural language processing function, question analysis function, candidate answer generation function, and answer inference/generation function of intelligent question answering system.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T F.746.11	2020-08-13	16	11.1002/1000/14328

Keywords

Natural language processing, QA intelligent system, QA interfaces QA metadata, question answering.

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T F.746.11

Interfaces for intelligent question answering system

1 Scope

This Recommendation addresses the descriptions for interfaces among modules and functions related to intelligent question answering systems. In particular, the scope of this Recommendation includes interfaces and functional features for the following modules of the intelligent question answering system:

– Natural language processing,

– Question analysis,

– Candidate answer generation,

– Answer inference/generation.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T F.746.3] Recommendation ITU-T F.746.3 (2015), *Intelligent question answering service framework*.

[ITU-T F.746.7] Recommendation ITU-T F.746.7 (2018), *Metadata for an intelligent question answering service*.

[ITU-T H.703] Recommendation ITU-T H.703 (2016), *Enhanced user interface framework for IPTV terminal devices.*

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 named entity recognition [ITU-T F.746.3]: A function that recognizes named entities such as PLO which are people, locations and organizations from the sentences. The PLO can be decomposed into more specific named entities depending on the applications.

3.1.2 natural language processing [ITU-T F.746.3]: A method that analyses text in natural languages through several processes such as part-of- speech recognition, syntactic analysis and semantic analysis.

3.1.3 semantic analysis [ITU-T F.746.3]: A function that recognizes the semantic relations among the words around predicates that exist in the same sentence. The semantic analysis function then generates a semantic predicate argument structure (PAS).

3.1.4 speech [ITU-T H.703]: Speech is the vocalized form of human communication.

3.1.5 syntactic analysis [ITU-T F.746.3]: A function that analyses sentence structures and generates dependency relations among words based on dependency grammars.

3.1.6 knowledge base [ITU-T F.746.7]: A collection of knowledge resources that consist of structured and unstructured data. The knowledge base is used to provide information to the various applications that are related to information provisioning such as QA systems and search systems.

3.1.7 question answering [ITU-T F.746.7]: A system that provides answers in a natural language to questions which are in the natural language form by analysing the questions and all the knowledge resources that are available to the system.

3.2 Terms defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

DB	Data Base
ID	Identifier
IQAS	Intelligent Question Answering Service
IR	Information Retrieval
KB	Knowledge Base
LAT	Lexical Answer Type
NE	Named Entity
NLP	Natural Language Processing
NNG	Noun General
PAS	Predicate Argument Structure
PLO	People, Location, Organization
POS	Part of Speech
QA	Question Answering
SAT	Semantic Answer Type
UIMA	Unstructured Information Management Architecture
VV	Verb
XML	extensible Mark-up Language

5 Conventions

None.

6 Overview

Intelligent question answering (QA) system is an advanced function to generate answers for the user's question in a natural language.

Figure 1 presents the exemplary QA architecture. The QA system consists of several functional blocks: Natural language processing, question analysis, candidate answer generation, answer inference and generation functional blocks.

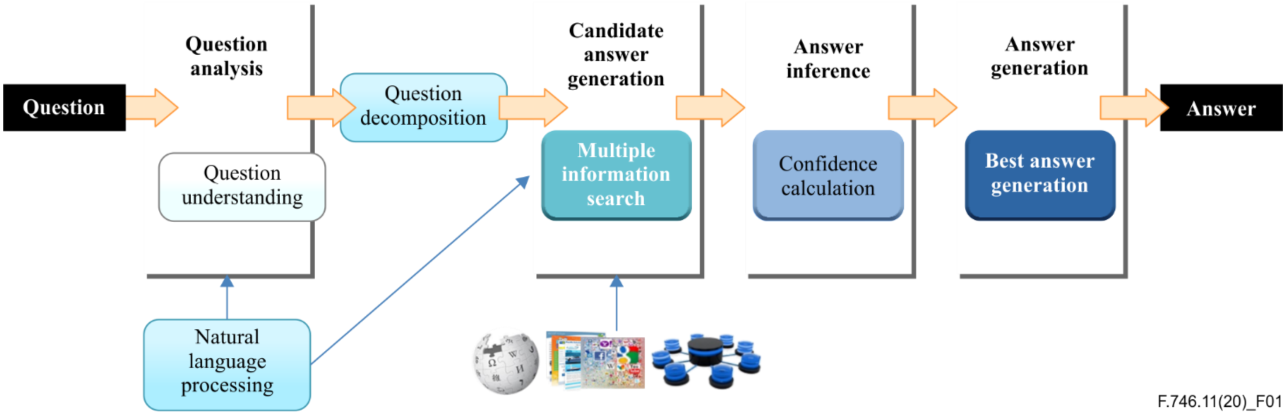

Figure 1 – Example of a QA architecture [ITU-T F.746.3]

This Recommendation addresses interfaces among modules and functions related to intelligent question answering services to present details of data and operation functions. [ITU-T F.746.3] and [ITU-T F.746.7] have specified some elements and metadata that are applicable to intelligent question answering services.

This Recommendation selects basic interfaces among modules and functions from these specifications that are applicable to intelligent question answering services. Names of elements/attributes and functional entities are quoted as they are in the specifications, to distinguish the relationship among the standards.

7 Functional components of intelligent question answering service

[ITU-T F.746.7] gives the definition for major functional components for intelligent question answering service (IQAS). IQAS functional components are basically composed of natural language processing, question analysis, candidate answer generation, answer inference and answer generation functional blocks as follows:

– **Natural language processing functional block**, which supports natural language processing on the QA server,

– **Question analysis functional block**, which supports question analysis on the terminal,

– **Candidate answer generation functional block**, which supports candidate answer generation by searching information in various data bases (DBs),

– **Answer inference/generation functional block**, which supports answer inference based on feature normalization and ranking of candidate answers and best answer generation on the terminal.

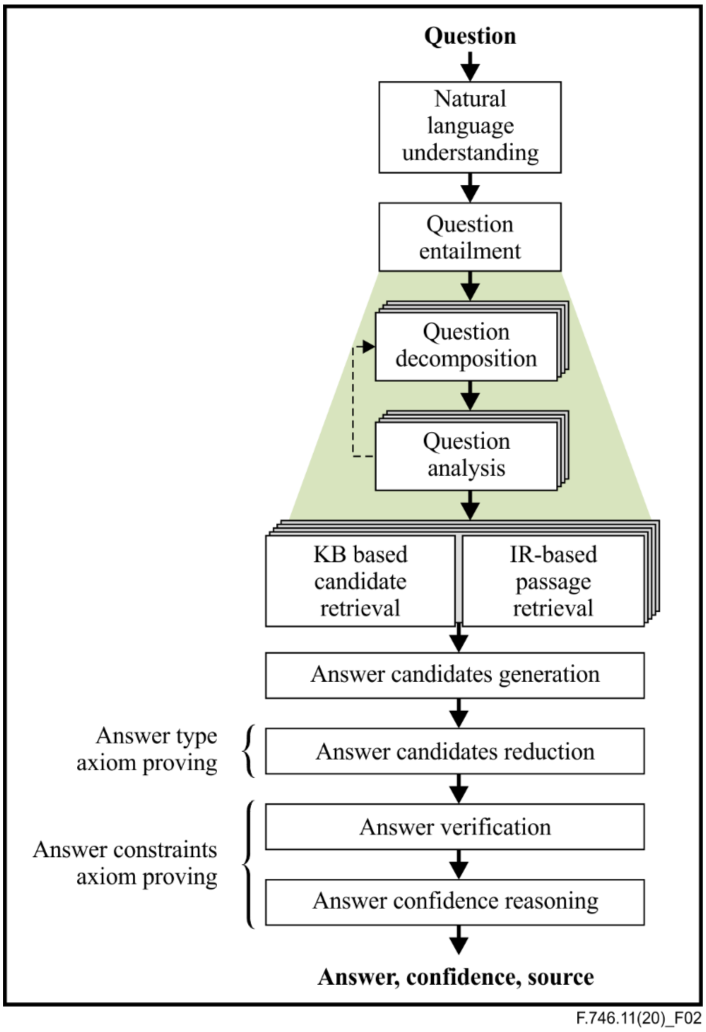
Question
Natural language understanding
Question entailment
Question decomposition
Question analysis
KB based candidate retrieval
IR-based passage retrieval
Answer candidates generation
Answer type axiom proving
Answer candidates reduction
Answer constraints axiom proving
Answer verification
Answer confidence reasoning
Answer, confidence, source
F.746.11(20)_F02

Figure 2 – Information flow of functional blocks in IQAS system [ITU-T F.746.7]

8 Interfaces for natural language processing

This clause specifies interfaces for natural language processing (NLP) function. The NLP module consists of part of speech (POS) analysis, named entity (NE) analysis, dependency analysis, semantic role labelling and ellipsis recovery. The natural language processing function analyses user's questions as well as target documents for knowledge extraction using the semantic and syntactic analysis technology.

The information flow of natural language processing is described in Figure 3 as sequentially connected submodules which analyse input documents to generate one best main result for each processing sub-module.

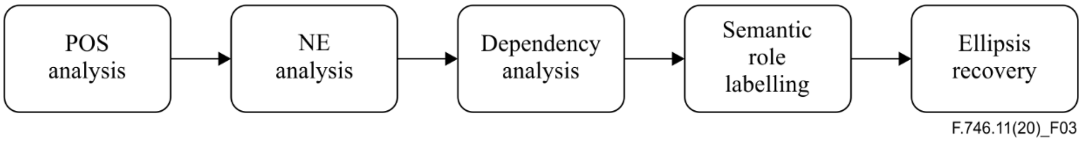

Figure 3 – Information flow of natural language processing

Table 8-1 illustrates the functional description and summarizes the input/output of each submodule for the natural language processing module.

Table 8-1 – Interfaces of functional blocks for natural language processing module

Functional blocks	Function description	Input	Output
POS analysis	To recognize parts of speech in the sentences and assign relevant POS tags considering contextual meaning of the target sentences.	Input sentence	POS tagged sentence in N-doc structure
NE analysis	To recognize named entities such as people, locations, organizations (PLO) and others from the sentences.	POS tagged sentence in N-doc	NE tagged sentence in N-doc structure
Dependency analysis	To analyse sentence structures and generate dependency relations among words based on dependency grammars.	POS/NE tagged sentence in N-doc	Dependency marked sentence components in N-doc structure
Semantic role labelling	To recognize the semantic relations among the words around predicates that exist in the same sentence and assign roles to the analysed semantic entities.	N-doc marked with POS/NE/Dependency	Semantic roles for each component stored in N-doc structure
Ellipsis recovery	To detect the ellipsis in a sentence and replace them with relevant noun phrases for the input sentences.	N-doc with previous analysis results	N-doc filled with recovered entities as a list

The boxes below present example class API for the POS analysis submodule. The input is a structure called "N_doc structure" of the input sentence. The output of this module is the unit of words with assigned POS tags.

The input of the POS analysis submodule is exemplified in the following box.

```
{
    "sentence": [
        {
            "id": 0,
            "text": "Olympic Tug of war is an Olympic program that pits two
teams against each other in the Olympic games ",             "morp": [],
            "word": []
        }
    ]
}
```

Output of the POS analysis is exemplified in the following box. The output is the N-Doc structure which stores the words and results of the POS analysis.

```
{
    "sentence": [
        {
            "id": 0,
            "text": "Tug of war is an Olympic program that pits two teams
                  against each other in the Olympic games.",
            "morp": [
                {"id": 0, "lemma": "Olympic", "type": "NNG", "position": 0,
                  "weight": 0.9},
                {"id": 1, "lemma": " Tug of war", "type": "NNG", "position":
                  10, "weight": 0.0701606},
                {"id": 2, "lemma": "in", "type": "JKB", "position": 22,
                  "weight": 0.0332677},
                {"id": 3, "lemma": " Olympic", "type": "NNG", "position": 26,
                  "weight": 0.9},
                {"id": 4, "lemma": "an", "type": "JKB", "position": 35,
                  "weight": 0.153407},
                {"id": 5, "lemma": "program", "type": "NNG", "position": 42,
                  "weight": 0.0789692},
                {"id": 6, "lemma": "pits", "type": "VV", "position": 54,
                  "weight": 0.0822907},
                {"id": 7, "lemma": "teams", "type": "NNG", "position": 58,
                  "weight": 0.152575},
                {"id": 8, "lemma": "against", "type": "JKO", "position": 64,
                  "weight": 0.137686},
                {"id": 9, "lemma": "each", "type": "VV", "position": 68,
                  "weight": 0.9},
                {"id": 10, "lemma": "other", "type": "ETM", "position": 74,
                  "weight": 0.184941},
                {"id": 11, "lemma": "Olympic", "type": "NNG", "position": 78,
                  "weight": 0.9},
                {"id": 12, "lemma": "games", "type": "NNG", "position": 88,
                  "weight": 0.135556}, ],
            "word": [
                {"id": 0, "text": "Olympic""type": "", "begin": 0, "end": 0},
                {"id": 1, "text": " Tug of war", "type": "", "begin": 1,
                  "end": 2},
                {"id": 2, "text": "Olympic", "type": "", "begin": 3, "end":
                  4},
                {"id": 3, "text": "Program", "type": "", "begin": 5, "end":
                  6},
                {"id": 4, "text": "pits", "type": "", "begin": 7, "end": 8},
                {"id": 5, "text": "teams", "type": "", "begin": 9, "end": 10},
                {"id": 6, "text": "Olympic", "type": "", "begin": 11, "end":
                  11},
                {"id": 7, "text": " games", "type": "", "begin": 12, "end":
                  12},            ]
        }
    ]
}
```

9 Interfaces for question analysis

This clause specifies interfaces for a user question analysis function. The question analysis module as shown in Figure 4 analyses questions which are input in a natural language by the user, understands the user's intention, and recognizes various information on the answers that should be presented as the output of the intelligent question answering system.

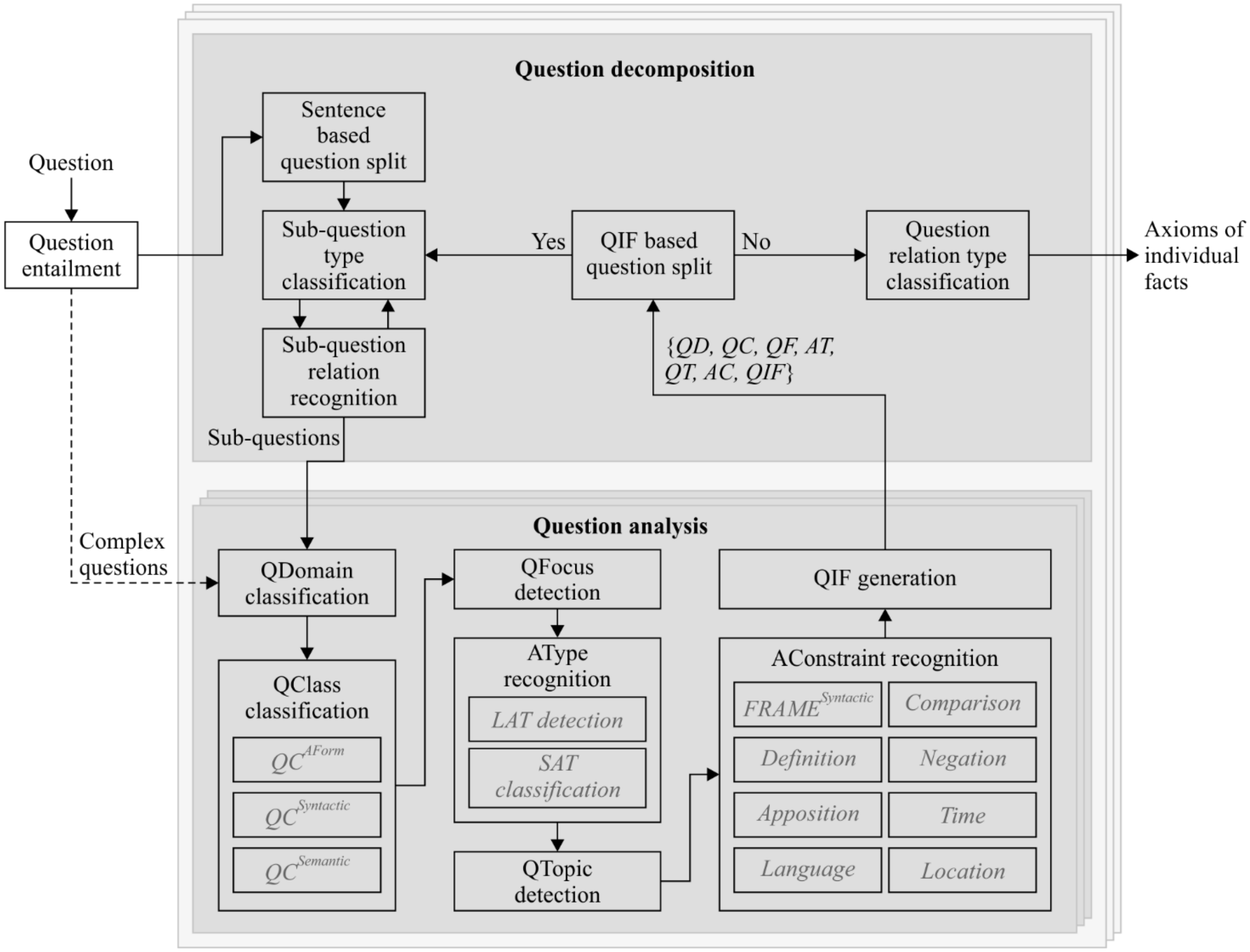

Figure 4 – Sub-modules for question analysis [ITU-T F.746.7]

The question analysis module consists of the following functional blocks:

- Question decomposition and recognition of sub-question relations,
- Question type classification,
- Recognition of question focus, answer type, and reliability,
- Question topic detection.

Table 9-1 describes the functional description and summarises the input/output of each functional block for the question analysis module.

Table 9-1 – Interfaces functional blocks for question analysis module

Functional blocks	Function description	Input	Output
Question decomposition and recognition of sub-question relations	To decompose a question based on its sentence structure and meaning. To recognize the decomposed sub-question types and relations among them.	Question focus, lexical answer type (LAT), NLP-analysed question	Sub question information and relation information
Question type classification	To define question types based on QA strategies depending on the domain. To classify the question types.	Question, NLP-analysed question	Pre-defined question types
Recognition of question focus, answer type, and reliability	To recognize question focus for the words or phrases which can be replaced by answer candidates. To assign the required answer types for lexical answer types (LAT) and semantic answer types (SAT).	NLP-analysed question, rule dictionary for answer type recognition	Recognized question focus and answer type (LAT and SAT)
Question topic detection	To recognize major entities in the question sentence and detect the core topic of the question.	Question, NLP-analysed question, Title list of on-line encyclopaedias	Core topic of question

The functional block "Question decomposition and recognition of sub-question relations" divides a question based on its sentence structure and meaning and does the function of recognizing sub-question types and relations among them. To do this decomposition, question decomposition types are pre-defined as shown in Table 9-2.

Table 9-2 – Question decomposition types

Decomposition types	Description
QD_Pa	Sub-questions are in parallel relations.
QD_Pa_Se	Sub-questions are separate and in parallel relations.
QD_Ne	Sub-questions are in nested relations. One question is nested in another question.
QD_Ne_Se	Sub-questions are in nested relations. They are separate questions.
QD_None	The question is not decomposable.
QD_None_Se	The non-decomposable questions are separated.

The following text presents example class API for a functional block "Question decomposition and recognition of sub-question relations". The input is N_doc structure of user's question and LAT vector of the original question. The output of this block is the sub-question relation vector and question decomposition information vector.

Class API for question analysis module is presented in clause A.1.

10 Interfaces for candidate answer generation

This clause specifies interfaces for a candidate answer generation function. A candidate answer generation module performs the index and search functions on the document collections and generates candidate answers from the input query using search results and various databases such as a structured/unstructured knowledge base as described in Figures 5 and 6.

The answer candidate generation block generates all possible answer candidates from the structured/unstructured resources based on the question and question division information.

The answer linguistic axiom prover block selects candidate answers for the evidence collection target through answer type and answer constraint axiom proving.

The answer evidence retrieval and contextual axiom prover collects evidences for the answers and verifies the axiom for the contexts.

Class API for candidate answer generation is found in clause A.2.

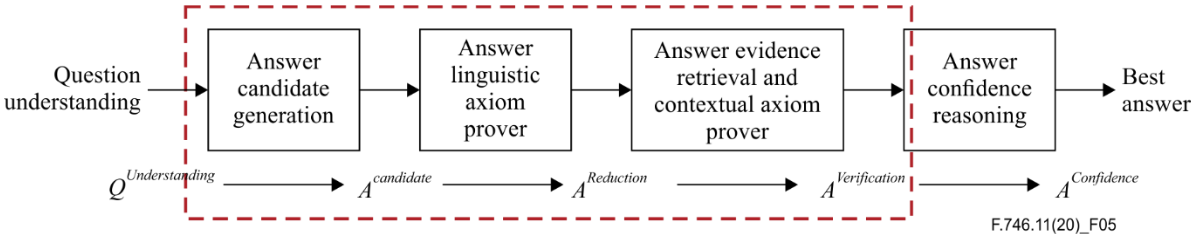
Question understanding
Answer candidate generation
Answer linguistic axiom prover
Answer evidence retrieval and contextual axiom prover
Answer confidence reasoning
Best answer
$Q^{Understanding}$
$A^{candidate}$
$A^{Reduction}$
$A^{Verification}$
$A^{Confidence}$
F.746.11(20)_F05

Figure 5 – Sub-blocks for the candidate answer generation module

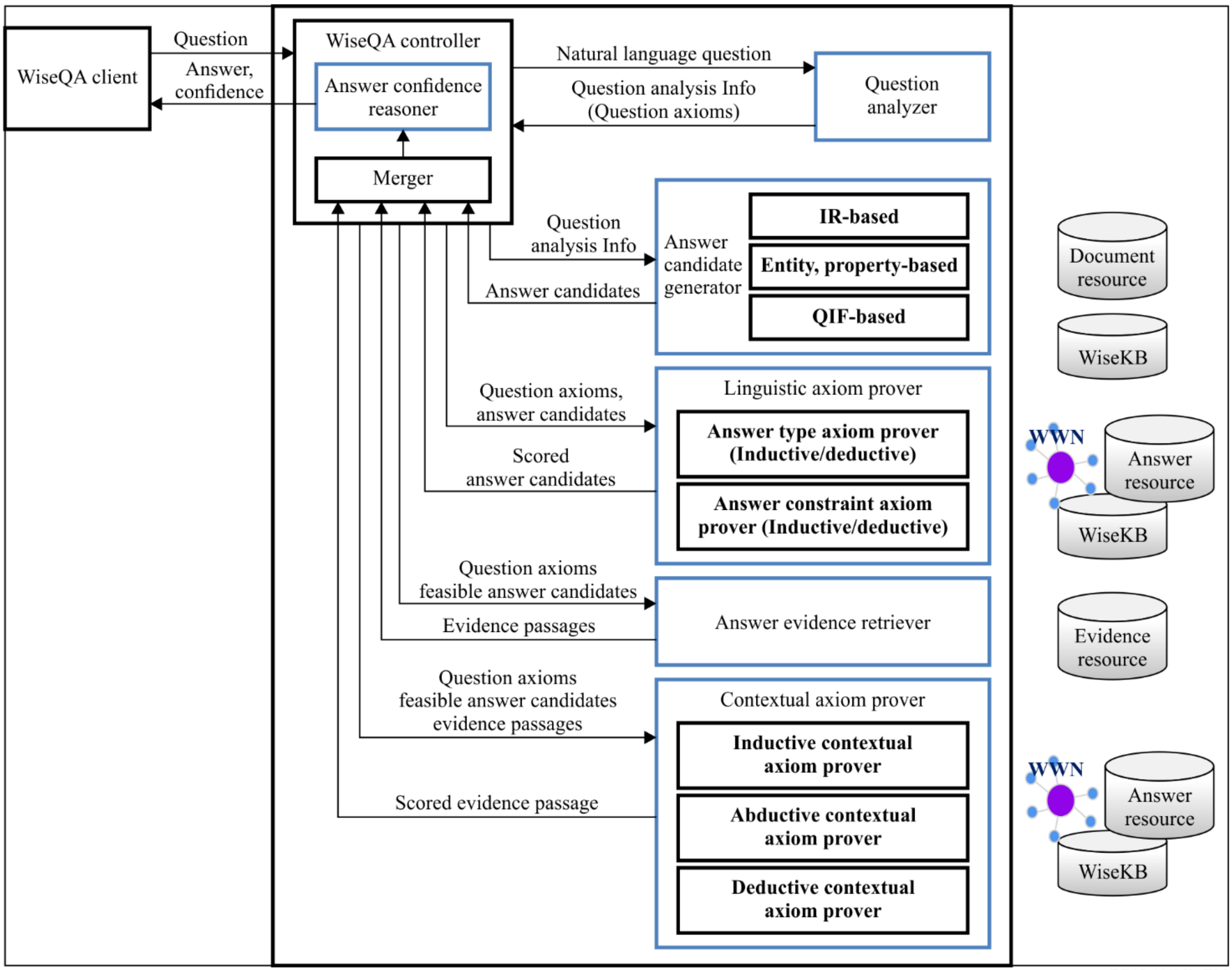
WiseQA client
Question
Answer, confidence
WiseQA controller
Answer confidence reasoner
Merger
Natural language question
Question analysis Info (Question axioms)
Question analyzer
Question analysis Info
Answer candidates
Answer candidate generator
IR-based
Entity, property-based
QIF-based
Document resource
WiseKB
Question axioms, answer candidates
Scored answer candidates
Linguistic axiom prover
Answer type axiom prover (Inductive/deductive)
Answer constraint axiom prover (Inductive/deductive)
WWN
Answer resource
WiseKB
Question axioms feasible answer candidates
Evidence passages
Answer evidence retriever
Evidence resource
Question axioms feasible answer candidates evidence passages
Scored evidence passage
Contextual axiom prover
Inductive contextual axiom prover
Abductive contextual axiom prover
Deductive contextual axiom prover
WWN
Answer resource
WiseKB
F.746.11(20)_F06

Figure 6 – Candidate answer generation module in QA system

11 Interfaces for answer inference/generation

This clause specifies interfaces for candidate answer generation function. The function of answer inference and the generation module is to decide and generate the best answer by measuring reliabilities of the answer candidates using the query axiom, the filtered answer candidates, and the reasoned answer candidates as features to determine ranks of the answer candidates, based on the calculated reliability. The answer candidates are filtered and reasoned out based on a similarity between the query axiom and the answer candidates and by using an inductive, deductive or an abductive reasoning.

The inference and generation modules compare a threshold value with a reliability ratio of a best answer candidate to the second-best answer candidate, readjusting the determined ranks according to the result of the comparison, and detecting the best answer candidate, determined through the readjustment, as a final answer of the question answering service.

Class APIs for answer inference/generation are presented in clause A.3.

Table 11-1 – Interfaces of functional blocks for answer inference/generation module

Functional blocks	Function description	Input	Output
Reliability measuring	To measure reliabilities of the answer candidates using the query axiom, the filtered answer candidates, the reasoned answer candidates. To determine ranks of the answer candidates, based on the calculated reliability.	Top N answer candidates Evidence feature vector	Ranked list of Top N answer candidates Confidence Source information
Best answer generation	To compare a threshold value of a best answer candidate to the second-best answer candidate. To readjust the determined ranks according to a result of the comparison. To generate the best answer candidate as the final answer.	Ranked list of Top N answer candidates Confidence Source information Data structure of the answer candidates	If threshold met, best answer produced, otherwise no answer

Annex A

Class API for intelligent question answering system

(This annex forms an integral part of this Recommendation.)

This annex describes class API for intelligent question answering system for each module.

A.1 Class API for question analysis

A.1.1 Class API for QDecomposition

This clause provides information for the class API for QDecomposition.

Class Name
QDecomposition
Description
Natural language question decomposition, recognition of decomposition type, sub-question type, sub-question relations
Include files

```
#include "QAnalHeader/QDecomposition.h"
#include "QAnalHeader/NDocUtil.h"
#include "QAnalHeader/RegEx.h"
#include "QAnalHeader/QStruct.h"
#include "QAnalHeader/QAnalRSCMng.h"
```

Member Variables

```
/**
Question Decomposition Type)
*/
typedef enum _Question_Decomposition_Type_{
     QD_Pa,             /**questions in parallel*/
     QD_Ne,             /**nested questions*/
     QD_None,      /**<not decomposable*/
} eQDType;

/**
SubQuestion Type
*/
typedef enum _SubQuestion_Type_{
     SQT_Fact, /**<fact type*/
     SQT_RelFact,  /**<related fact type (association inference type)*/
     SQT_Question, /**<question type*/
     SQT_InnerQ,   /**<Nested, Inner Question*/
     SQT_OuterQ    /**<Nested, outer Question*/
} eSQType;

/**
Relation of the sub-questions
*/
typedef enum _SubQ_Relations_
{
     SQR_And,      /**<and relation*/
     SQR_Dep,      /**<dependency relation*/
     SQR_None,     /**<no relation*/
     SQR_SUPPORT        /**<answer constraint relation*/
}eSQRelation;

/**
```

```
doc structure to store information about the sub-questions
*/
typedef struct _QDecomposition_Info_
{
     unsigned int iID;/**<SubQ ID*/
     string strSubQ;/**<SubQ string*/
     int iStartPos;/**<start position of the original question*/
     int iEndPos;/**< end position of the original question */
     eSQType subQType;/**<SubQ Type*/
     string strSubQType; /**<SubQ Type string*/
     sQAnal_Unit qAnalUnit;/**<analysis result of SubQ */
     FrameStruct qSFrame;/**<question semantic frame list*/
}QDecomp_Info;

/**
Sub-question relation structure
*/
typedef struct _SubQ_Relation_Triple_
{
     unsigned int iID1;/**<SubQ ID1*/
     unsigned int iID2;/**<SubQ ID2*/
     eSQRelation relation;/**<relation between SubQ ID1and ID2 */
     string strRelation; /**< relation string between SubQ ID1 and ID2 */
}sSQ_Relation;

/**
Analysis structure for the sub question
*/
typedef struct _QAnal_Info_
{
     string strOrgQuestion;/**<question string*/
     sQAnal_Unit orgQUnit;/**<info on original question*/
     eQDType qDecompType;/**<sub-question type*/
     vector<sSQ_Relation> vSubQRelation;/**<relation information*/
     vector<QDecomp_Info> vSubQInfo;/**<information on the sub-question*/
     double dQH_Weight;/**<question assumption reliability*/
}sQAnal_Info;

private:
     CQAnalRSCMng *pRscMng;/**<question resource manager handle*/
```

Member Functions

```
public:
     CQDecomposition(void);
     ~CQDecomposition(void);
private:
     vector<QDecomp_Info> Split_Question(sQAnal_Unit &sQAnal_Info);
/**<function to divide into sub-questions */
void SubQuestions_Classifier(vector<QDecomp_Info> &vSubQs, sQAnal_Unit
sQAnal_Info); /**<function to classify sub-questions into question types */
     eQDType Recognize_SubQ_Relations(vector<QDecomp_Info> vSubQs,
vector<sSQ_Relation> &vSubQRel); /**<function to recognize relations between
sub-questions */
     vector<unsigned int> GetSubQSentIDs(sQAnal_Unit sQAnal_Info);
     bool IsQuesitonByRule(string strTaggedSent); /**<function to decide if
the question is detected based on the question pattern*/
     vector<sQF_Info> GetQFsInSubQ(QDecomp_Info sSubQ, vector<sQF_Info>
vOrgQFInfo); /**<function to get question focus from sub-questions */
vector<sLAT_Info> GetLATsInSubQ(QDecomp_Info sSubQ, vector<sLAT_Info>
vOrgLATInfo); /**<function to get question lextical answer type (LAT) from
sub-questions */
```

A.1.2 Class_API for question type classification

This clause provides information for the Class_API for question type classification.

Class Name	
CQClassifier	
Description	
Core class for classifying questions	
Include files	

#include "QAnalHeader/ Classifier_Rules.h"
#include "QAnalHeader/ Classifier_ML.h"
#include "QAnalHeader/ QAnalRSCMng.h"	

Member Variables	
protected: CQAnalRSCMng *pRscMng;/**<Resource manager handle*/	
Member Functions	

public:
     /** fuction for registering Resource manager handle */
     virtual void SetRscMng(CQAnalRSCMng * pRscManager) = 0;
 	
     /**function for recognizing question types */
     virtual vector<sQAnal_CQT> QClassifer(string strFeatures, string strTaggedQ, bool bIsMC, bool bIsCB);
 	
     /**feature extraction function for machine learning Classification */
     virtual string ExtractFeatures(string strQ, N_Doc ndoc);
 	
protected:
     /**Function for a hybrid Lexico-Semantic Rule and machine learning result */
     vector<sQAnal_CQT> HybridCQT(vector<sQAnal_CQT> vRuleCQTs, vector<sQAnal_CQT> vMLCQTs);	

Class Name	
CQClassifier_Rules	
Description	
Core class for classifying questions by rule-based method	
Include files	

#include "QAnalHeader/Classifier_Rules.h"
#include "QAnalHeader/QAnalRSCMng.h"
#include "QAnalHeader/RegEx.h"	

Member Variables	
private: CQAnalRSCMng *pRscMng; vector<sCQTRules> vRules; /**rule dictionary for classification*/	
Member Functions	
public: CClassifier_Rules(void); ~CClassifier_Rules(void); protected: /** Registration for the resource handler and classification rules */	

```
     void SetRscMng_Rules(CQAnalRSCMng *pRscManager, vector<sCQTRules>
&vClassifier_Rules);

     /** classification function for rule-based function */
     virtual vector<sQAnal_CQT> Classifier_Rules(string strTaggedQ, bool
bIsMC, bool bIsCB);

private:
     /** function to find question classification candidates based on rules */
     vector<sQAnal_CQT> GetCQTCandidates(string strTaggedQ, bool bIsMC, bool
bIsCB);

     /** function to integrate classified question types based on rules */
     void UnifyingCandidates(vector<sQAnal_CQT> &vCandidates);
```

Class Name

CQClassifier_ML

Description

Core class for classifying questions by machine learning method

Include files

```
#include "QAnalHeader/Classifier_ML.h"
#include "QAnalHeader/QAnalRSCMng.h"
```

Member Variables

private:

    CQAnalRSCMng *pRscMng;/

    CRF_MODEL *pCRFHandle;/**<QAT machine learning model instance */

Member Functions

```
public:
     CClassifier_ML(void);
     ~CClassifier_ML(void);

protected:
     /** Registration for the resource handler and classification ML    */
     void SetRscMng_ML(CQAnalRSCMng * pRscManager, CRF_MODEL *crf_model);

     /** classification function based on ML */
     virtual vector<sQAnal_CQT> Classifier_ML(string strFeatures);

public:

     /** function for CQT classification based on features */
     void ClassifyCQT(string strFeature, vector<sQAnal_CQT> &vReturnCQT);

     /** function to extract QT features for ML */
     string ExtractQTFeature(N_Doc ndoc);
     string ExtractBiGramFeature(N_Doc ndoc);
     string ExtractSymbolFeature(N_Doc ndoc);
     string ExtractLastWordFeature(N_Doc ndoc);
```

Class Name

CQClassifier_AForm

Description

Question classification class according to the answer type

Include files

```
#include "QAnalHeader/QClassifier.h"
```

```
#include "QAnalHeader/QAnalRSCMng.h"
#include "QAnalHeader/QClassifier_AFrom.h"
```

Member Variables	
Member Functions	

```
public:
     CQClassifier_AForm(void);
     ~CQClassifier_AForm(void);

     void SetRscMng(CQAnalRSCMng * pRscManager);

     /** function to recognize CQT */
     sQAnal_CQT QClassifer(string strQ, N_Doc ndoc, string strTaggedQ, bool
bIsMC, bool bIsCB);
```

Class Name	
CQClassifier_Sem	
Description	
Question classification class based on semantic feature of the question	
Include files	

```
#include "QAnalHeader/QClassifier.h"
#include "QAnalHeader/QAnalRSCMng.h"
#include "QAnalHeader/QClassifier_sem.h"
```

Member Variables	
Member Functions	

```
public:
     CQClassifier_Sem(void);
     ~CQClassifier_Sem(void);

     void SetRscMng(CQAnalRSCMng * pRscManager);

     /** function to recognize question types (CQT) for questions */
     vector<sQAnal_CQT> QClassifer(string strQ, N_Doc ndoc, string strTaggedQ,
bool bIsMC, bool bIsCB);

     /** function to map the question classification results onto string */
     void MappingStrQType(vector<sQAnal_CQT> &vResult);
```

A.1.3 Class_API for recognition of question focus, answer type and reliability

This clause provides information for the Class_API for recognition of question focus, answer type and reliability.

Class Name	
CQAnalLAT	
Description	
Lexical answer type recognition module Class	
Include files	

```
#include "QAnalHeader/QAnalRSCMng.h"
#include "QAnalHeader/QAnalLAT_Rules.h"
#include "QAnalHeader/QAnalLAT_ML.h"
```

Member Variables	

```
CQAnalRSCMng *pRscMng;

//SAT recognition based on rules-module handle
```

```
CQAnalLAT_Rules *pRuleLAT;

// SAT recognition based on ML-module handle
CQAnalLAT_ML *pMLLAT;
```

Member Functions

```
     void SetRscMng(CQAnalRSCMng * pRscManager);

     /** function to recognize lexical answer types for a question */
     vector<sLAT_Info> RecognizeLAT(string strQ, N_Doc ndoc, vector<sQT_Info>
vQTs, string strTaggedQ);

     /** to get the LAT analysis handle based on rule-based method */
     CQAnalLAT_ML *GetQAnalLAT_ML_Handle() { return this->pMLLAT; };

protected:

     /** function to hybrid Lexico-Semantic Rule and ML results */
     vector<sLAT_Info> HybridLAT(vector<sLAT_Info> vRuleLATs,
vector<sLAT_Info> vMLLATs);

private:
     void EraseStopWordInLATs(vector<sLAT_Info> &vReturnLAT);
```

Function Name
RecognizeLAT
Class Name
CQAnalLAT
Description
Function to recognize lexical answer type for a given question
Syntax
vector<sLAT_Info> RecognizeLAT(string strQ, N_Doc ndoc, vector<sQT_Info> vQTs, string strTaggedQ);
Return Value
`vector<sLAT_Info> - Recognized LAT candidates vector`
Parameters
@param string strQ – original question string @param N_Doc d – language analysis result of the question @param vector<sQT_Info> vQTs – question type information recognized in a question @param string strTaggedQ – input string for rule matching

Function Name
GetQAnalLAT_ML_Handle
Class Name
CQAnalLAT
Description
Function to get the handle of ML based LAT results
Syntax
CQAnalLAT_ML *GetQAnalLAT_ML_Handle();
Return Value
`CQAnalLAT_ML *`
Parameters

Function Name
HybridLAT
Class Name
CQAnalLAT
Description
Hybrid function of Lexico-Semantic Rule and ML results
Syntax
vector<sLAT_Info> HybridLAT(vector<sLAT_Info> vRuleLATs, vector<sLAT_Info> vMLLATs);
Return Value
`vector<sLAT_Info>`
Parameters
@param vector<sLAT_Info> vRuleLATs – rule-based LAT result @param vector<sLAT_Info> vMLLATs – ML based LAT result

Class Name
CQAnalSAT
Description
SAT recognition class
Include files

`#include "QAnalHeader/QAnalRSCMng.h"`
`#include "QAnalHeader/QAnalSAT_Rules.h"`
`#include "QAnalHeader/QAnalSAT_ML.h"`

Member Variables

`CQAnalRSCMng *pRscMng;`
`CQAnalSAT_Rules *pRuleSAT;`
`CQAnalSAT_ML *pMLSAT;`

Member Functions

```
     void SetRscMng(CQAnalRSCMng * pRscManager);

     /** function to recognize Semantic answer type for a question*/
     vector<sSAT_Info> RecognizeSAT(string strQ, N_Doc ndoc, vector<sQT_Info>
vQTs, string strTaggedQ);

     /** function to get SAT analysis handle based on ML */
     CQAnalSAT_ML *GetQAnalSAT_ML_Handle() { return this->pMLSAT; };

     /** function to get SAT analysis handle based on rules */
     CQAnalSAT_Rules *GetQAnalSAT_Rule_Handle() { return this->pRuleSAT; };

protected:
     /** function to hybrid Lexico-Semantic Rule and ML results */
     vector<sSAT_Info> HybridSAT(vector<sSAT_Info> vRuleSATs,
vector<sSAT_Info> vMLSATs);

     /**function to expand the recognized SAT */
     void ExpansionSAT(vector<sSAT_Info> &vSATs);
```

Function Name
Class Name
CQAnalSAT
Description
Function to recognize SAT for a question
Syntax
vector<sSAT_Info> RecognizeSAT(string strQ, N_Doc ndoc, vector<sQT_Info> vQTs, string strTaggedQ);
Return Value
`vector<sSAT_Info> - recognized SAT results`
Parameters
@param string strQ – question string @param N_Doc d – language analysis result of the question @param vector<sQT_Info> vQTs – answer type information recognized in the question @param string strTaggedQ

Function Name
HybridSAT(
Class Name
CQAnalSAT
Description
Hybrid function of Lexico-Semantic Rule and ML results
Syntax
vector<sSAT_Info> HybridSAT(vector<sSAT_Info> vRuleSATs, vector<sSAT_Info> vMLSATs);
Return Value
`vector<sSAT_Info> - Hybrid SAT candidate result bector`
Parameters
@param vector<sSAT_Info> vRuleSATs – SAT results by rule-based mehod @param vector<sSAT_Info> vMLSATs – SAT results by ML-based mehod

Function Name
ExpansionSAT
Class Name
CQAnalSAT
Description
This function is to expand the SAT which was recognized.
Syntax
void ExpansionSAT(vector<sSAT_Info> &vSATs);
Return Value
`void`
Parameters
@param vector<sSAT_Info> &vSATs – Results of the recognized SAT

A.1.4 Class API for question topic detection

This clause provides information for the class API for question topic detection.

Class Name
CQAnalKeywords
Description
Function to recognize main keywords in a question
Include files
#include "QAnalHeader/QAnalRSCMng.h"
Member Variables
CQAnalRSCMng *pRscMng;
Member Functions

void SetRscMng(CQAnalRSCMng * pRscManager) { this->pRscMng = pRscManager;};
CQAnalRSCMng *GetRscMng() { return this->pRscMng; };
 	
/** function to get NE tagged objects from the language analysis results */
map<string, sTitle_Info> GetNEWords(N_Doc ndoc);
 	
/** function to get chunk information from the language analysis results */
map<string, sTitle_Info> GetChunkWords(N_Doc ndoc, vector<string> vStrType);

Function Name
GetNEWords
Class Name
CQAnalKeywords
Description
function to get NE tagged objects from the language analysis results
Syntax
map<string, sTitle_Info> GetNEWords(N_Doc ndoc);
Return Value
map<string, sTitle_Info> - recognized object information (key : text_morp begin)
Parameters
@param N_Doc ndoc – language analysis results

Function Name
GetChunkWords
Class Name
CQAnalKeywords
Description
function to get chunk information from the language analysis results
Syntax
map<string, sTitle_Info> GetChunkWords(N_Doc ndoc, vector<string> vStrType)

Return Value

```
map<string, sTitle_Info> - recognized object information (key : text_morp
begin)
```

Parameters

@param N_Doc ndoc - language analysis results
@param vector<string> vStrChunkTypes – chunk types to get

Class Name

CQAnalWikiTitle

Description

Entity Linking class to map the keywords in the question into the WIKI titles to resolve the ambiguity

Include files

```
#include "QAnalHeader/QAnalRSCMng.h"
#include "QAnalHeader/QAnalKeywords.h"
```

Member Variables

CQAnalRSCMng *pRscMng;

Member Functions

```
/** function to recognize WIKI titles */
     vector<sTitle_Info> GetWikiTitles(string strQ, N_Doc ndoc);

private:

     /** function to recognize candidates for WIKI titles */
     vector<sTitle_Info> GetWikiTitleCandidates(N_Doc ndoc);

     /** function to resolve the WIKI title ambiguity */
     void DisambiguationTitle(string strQ, vector<sTitle_Info> &vTitles);

     /** function to find WIKI title from the candidates */
     void FindWikiTitles(sTitle_Info sCandidate, vector<sTitle_Info>
&vTitles);

     /** function to search for the string if it is in the WIKI title
dictionary */
     void LookupWikiDic(string title, vector<sEntity_Info> &vEntities);
```

Function Name

GetWikiTitles

Class Name

CQAnalWikiTitle

Description

Function to recognize WIKI titles

Syntax

vector<sTitle_Info> GetWikiTitles(string strQ, N_Doc ndoc);

Return Value

```
vector<sEntity_Info> - Wikititle information
```

Parameters

@param string strQ - Question
@param N_Doc ndoc – language analysis results

A.2 Class API for candidate answer generation

A.2.1 Class_API for candidate answer index and search

This clause provides information for the Class_API for candidate answer index and search.

Class Name

DIndexingMultiThread

Description

Unstructured indexing: top class

Include files

none

Member Variables

```
private Thread t;
private String threadName;
private String filePath;
String syntacticLexical;
String syntacticRelation;
String surficialRelation;
long start,end;
String hbaseTableName;
CloudSolrServer solrcloud;
```

Member Functions

```
DIndexingMultiThread(String name, String hbaseTableName, CloudSolrServer
solrcloud, String filePath)
public void run()
public void start()
public String
makeTerm2StringWithWhiteSpace(ArrayList<HashMap<Integer,ArrayList<String>>>
sentenceList)
```

Class Name

PrimarySearch.cpp

Description

Unstructured indexing: top class

Include files

```
typedef struct {
        string docid;   // document ID
        string domain;  // wiki, dictionary
        string type;    // document, section, definition, passage
        string rowkey;  // hbase key (for fetching language analysis results)
        string page_struct;     // doc title
        string description;     // information for the origin of doc
        double weight;  // doc weight
        int ranking;    // doc ranking
        int s_sentid;   // start sentence ID
        int e_sentid;   // end sentence ID
        vector<string> topic;   // topic of the doc
        vector<TERM_INFO> syntacticLexical;     // keyword info matched in the
search
        vector<TERM_INFO> syntacticRelation;    // keyword info matched in the
search
        vector<TERM_INFO> surficialSemanticRelation;    // keyword info
matched in the search
        vector<TERM_INFO> semanticRelation;     // keyword info matched in the
search
} DOC_RESULT_STRUCT;

//Primary Search integration
typedef struct {
        string  query;
        //==============================
        // save the search result
        //==============================
        vector<DOC_RESULT_STRUCT> doc_result_list;      // to store doc search
results
        vector<DOC_RESULT_STRUCT> def_result_list;      // to store definition
search results
        vector<DOC_RESULT_STRUCT> sec_result_list;      // to store section
search results
        vector<DOC_RESULT_STRUCT> psg_result_list;      // to store paragraph
search results
} doc_result;
```

Member Variables	//Skipped//

Member Functions

```
void hexconvert( char *text, unsigned char bytes[] );
string replaceAll(string str, string pattern, string replace);
string* strSplit(string strOrigin, string strTok);
PrimarySearch(string solrIP, string hbaseIP, string jarPath);
string p_q_generation(string collectionName, N_Doc ndoc);
string p_q_reGeneration(string collectionName, N_Doc ndoc, vector<string>
candidateAnswer);
string p_q_reGeneration(string collectionName, N_Doc ndoc, string
docid_sentid);
vector<string> assignPassageRange(string docid_sentid, int range);
PrimarySearch();
~PrimarySearch();
string p_search( string collectionName, string query, int max_cnt);
string p_search( string collectionName, string query, string title, int
max_cnt);
string getLang(string collectionName, string rowkey);
string getRowkey(string docid);
```

A.2.2 Class_API for candidate answer generation sub-block

This clause provides information for the Class_API for candidate answer generation sub-block.

Class Name

AnswerUnit

Description

Answer candidates and structure of representing evidences

Include files

```
class AnswerUnit
{
public:

     string answer;
     string answer_key;      //
     double tot_weight;      //engine matching weight(normalized reliability)

     int total_sent;
     list<AnswerSent> answer_sent;
     CIFeature ci_feat; //context independent features
};

class AnswerFeature
{
public:

     AnswerFeature(void);
     ~AnswerFeature(void);

     double TyCor_score; //TyCor feature
     double Ln_score; //IR+ voc similarity feature
     double SEM_score; //syntax and meaning feature
     double CONS_score; //question constraint feature

     double IR_score; //IRweight

     string TE;              //integration

     double confidence; //answer confidence

     string src_ENGINE; // source of engine

};
// answer candidate sentences for Answer Evidence
class AnswerSent
{
public:
     string answer;
     double weight;

     string docID;      //answer doc ID
     int sentID;   //answer sentence ID
     QADocType doc_type; //

     //canGen results
     string source;  // source (Content, Title, ...)
     double DIRweight;  // doc search weight
     int DIRsrcType;  // 0=Doc, 1=secion, 2=sentence

     string title;
     string sent;
     string url;
     string cpname;
     unsigned long long date=0;
};

//context independent features
class CIFeature
{
public:
     string type;  // types
     string NEType;  // NE type
     double inQ = 0;  // if included in question

     double CIweight;    //after feature engineering
     double TyCor_vec[TyCor_vec_SIZE];

     string CI_feature; //CI feature for training str
};
```

Class Name

CandidateGenerator

Description

Top class for answer candidate generation

Include files

```
#include "DR/DR.h"
#include "QStruct.h"
#include "DBSolution/DBSolution.h"
#include "GetNDocHeader/GetNDoc.h"
#include "TyCor/TyCor.h"
#include "LingCor/LingCor.h"
#include "SpaTempCor/SpaTempCor.h"
#include "set"

typedef struct CandidateAnswer {
     string text;  // text of candidate answers
     string type;  // type of candidate answers (NE, NP, ...)
     string NEType;  // NE type
     string source;  // source (Content, Title, ...)
     string sentenceText;  // text of the sentence
     string docID;  // extracted document ID
     string paraID;  // extracted paragraph ID
     int sentenceID = -1;  // extracted sentence ID
     int beginMorpID = -1;  // starting morpheme ID
     int endMorpID = -1;  // last morpheme ID
     int beginWordID = -1;  // strting word ID
     int endWordID = -1;  // last word ID
     LATScore LATScr;  // LAT similarity
     SATScore SATScr;  // SAT similarity
     SpaceScore spaceScr;  // space similarity
     TimeScore timeScr;  // time similarity
     LingScore lingScr;  // context similarity
     string dependency;  // used in TE
} CandidateAnswer;

typedef struct CandidateAnswerConfig {
     bool SUPPORT_PASSAGE_SEARCH=false;
     bool EXTRACT_NE=false;
     bool EXTRACT_NP=false;
     bool USE_LAT_SCORE=false;
     bool USE_SAT_SCORE=false;
     bool USE_SPACE_SCORE=false;
     bool USE_TIME_SCORE=false;
     bool USE_WORD_SCORE=false;
     bool USE_WORD_BI_SCORE=false;
     bool USE_WORD_ORDER_SCORE=false;
     bool USE_SYN_SCORE=false;
     bool VERBOSE_Q_LAT=false;
     bool VERBOSE_Q_DEPENDENCY=false;
     bool VERBOSE_DOCS_DEPENDENCY=false;
     bool VERBOSE_SYN_SCORE=false;
} CandidateAnswerConfig;

typedef pair<int,int> CandidateAnswerId;
```

Member Variables

```
     DBSolution *dbSol;
     string dbPath;
     CGetNDoc getNDoc;
     TyCor tyCor;
     LingCor lingCor;
     SpaTempCor spaTempCor;
     CandidateAnswerConfig config;
```

Member Functions

```
     CandidateGenerator();
     bool initialize( string rscDir, DBSolution *dbSol, string dbPath, 
Searcher *kornetSearcher=NULL );
     vector<CandidateAnswer> generateCandAns( QDecomp_Info& q, doc_result& dr 
);
     static void printCandidateAnswers( vector<CandidateAnswer>& cands );
     static void printCandidateAnswer( CandidateAnswer& cand, int index=-1 );
```

Class Name

AnswerHypo

Description

Extraction of answer candidate and hypothesis generation according to the questions

Include files

```
#include "math.h"

#include "QAConfig.h"
#include "AnswerType.h"

//===========Question analysis
#include "QStruct.h"
#include "QAnalHeader/QAnalyzer.h"
#include "QAJsonHeader/QAnalJsonRW.h"
//=========== document search
#include "PrimarySearch/PrimarySearch.h"
#include "QAJsonHeader/DocResultParser.h"
//=========== candidate answer generation
#include "CanGen/CanGenerator.h"
#include "CI/TypeCor.h"
/*/=========== KB search
#include "KBInterface/KBInterface.h"
//=========== real time extraction of candidate answers
#include "CanGen/CandidateGenerator.h"
/**/
/***********************************************/
//for multi-thread
extern CQAnalRSCMng * qResourceMng; //
extern Dictionary * gkornet;     //lexical semantic concept network

extern map<int, string> *ExAtMap;
extern map<string, int> *rExAtMap;

extern QAConfig QAConfig_core;
```

Member Variables

```
     QAnal_Result Q;              //question analysis result structure
     CQAnalJsonRW * QJsonRW; // question analyzer JsonRW

     //Primary Search------------------
     PrimarySearch * ps;
     DocResultParser *DocJsonRW;
     double DIR_top_weight;
     //anwer candidate extraction------------------
     float CONF_CUT_OFF;
     CanGenerator * CanGen;
     TypeCor * CI;
```

Member Functions

```
     //real time candidate answer extraction
     int get_answerCandidate_fromIR(string Q_json, int subQ_idx, doc_result 
lvResult, list<AnswerUnit> & RES);
     //subQ question merge
     bool   merge_answerCandidate_fromSubQ(vector<CandidateAnswer> 
sub_AnsCan_vec, int subQ_idx, map<string, AnswerUnit> & AnsUnit_map);
     //pre_softfilter
     int pre_soft_filter(CandidateAnswer & AnsCan, int subQ_idx);
```

Function Name
get_answerCandidate_fromIR
Class Name
AnswerHypo
Description
Candidate answer generation based on document search
Syntax
int get_answerCandidate_fromIR(string Q_json, int subQ_idx, doc_result lvResult, list<AnswerUnit> & RES);
Return Value

return int; // number of candidate answers
list<AnswerUnit> RES : //candidate answer list basically ordered

Parameters
string input_Json //result json (with "||" division) int subQ_idx: // processed question index (-1: original question)

Function Name
pre_soft_filter
Class Name
AnswerHypo
Description
Filtering function for the answer candidates as the doc search results
Syntax
int pre_soft_filter(CandidateAnswer & AnsCan, int subQ_idx);
Return Value

return int; //if filtering applied
0: not object for filtering
1: if more than 1, filtering should be applied

Parameters
CandidateAnswer & AnsCan // individual answer candidate int subQ_idx: (processed question) index (-1: original question)

A.3 Class API for answer inference/generation

This clause provides information for the class API for answer inference/generation.

Class Name

AnswerConfidence

Description

Among answer candidates from different answer generation module, to select the answer candidate with the highest reliability and to verify the answer to the question.

Include files

```
#if !defined(AFX_ANSWERRANK_H__C4797D53_04A5_4E4E_BE8F_841D4BC76032__INCLUDED_)
#define AFX_ANSWERRANK_H__C4797D53_04A5_4E4E_BE8F_841D4BC76032__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define INDI_MAX 30

#include "QAConfig.h"
```

Member Variables

```
xmlrpc_c::clientSimple myClient;
     /// Object pointer of QAConfing for answer inference resource loading and
reliability model loding
     QAConfig *qconfig;

     Merge * AnsMerge;

     /// answer filtering vector
     vector<string> except_answer;

     /// answer candidates integration exception rules
     vector<string> except_is_redirect_rule;

     /// answer filtering rule map
     map<string, string> filter_map;

     /// answer filtering rule map using LAT and SAT
     map<string, string> latsat_filter_map;
```

Member Functions

```
public:
     AnswerRank();
     virtual ~AnswerRank();
     void ARank_THREAD_init();
     string soft_filter_4UIMA(string json_in);
string answer_merge_4UIMA(string json_in);
private:
     xmlrpc_c::clientSimple myClient;
     QAConfig *qconfig;
     /// Merge object pointer for answer candidates integration
     Merge * AnsMerge;
     /// answer filtering vector
     vector<string> except_answer;
     /// space information NE vector
     vector<string> space;
     /// time information NE vector
     vector<string> time;
     map<string, string> filter_map;
     map<string, string> latsat_filter_map;
```

```
    bool latsat_filter(string NE_Type, map<string, string> & latsat_filter_map,
vector<sLAT_Info> latVec, string first_SAT, double KB_score, bool q_type);
    bool answer_filter(string NE_Type, map<string, string> & filter_map, string
SAT_Type);
    bool answer_filter_lat(string NE_Type, vector<sLAT_Info> latVec, bool
q_type);
    bool soft_filter_lat(string candidate, string NE_Type, string SAT_Type,
vector<sLAT_Info> latVec, double KB_score, bool q_type, string first_SAT);
    bool soft_filter_sat(string candidate, string NE_Type, string SAT_Type,
vector<sLAT_Info> latVec, double KB_score, bool q_type, string first_SAT);
    bool soft_filter_string(string candidate, string NE_Type, vector<sLAT_Info>
latVec, string question, double KB_score, bool q_type);
    bool update_resource();
    bool except_one_char_answer(string & answer, string & taggedQ, QAnal_Result
Q);
    string make_feature(list<AnswerUnit>::iterator lpAu, int rank);
    double conf_wiseqa(int blank_size, string answer_type, int q_exam_size,
list<AnswerUnit>::iterator lpAu, bool lat_flag, bool sat_flag, string q_type, int
rank, string SAT_CONF_TYPE);
    double conf_all(int blank_size, string answer_type, int q_exam_size,
list<AnswerUnit>::iterator lpAu, string q_type, int rank, string SAT_CONF_TYPE);
/** Reliability calculation */
    bool overlap_question(vector<string> vec, string taggedQ, string question,
QAnal_Result Q);
    /** final answer inference */
    int re_ranking_answerCandidate(list<AnswerUnit> & RES, int mode, bool
select, int select_size, bool negation, bool lat_flag, bool sat_flag, string
qstring, string SAT_Type, QAnal_Result Q, string q_type, string SAT_CONF_TYPE,
string first_SAT);
    bool exist_kb_AnswerUnit(list<AnswerUnit> & RES, string json);
    double thre_type(string conf_type, bool lat, bool sat); //threshold measure
```

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	**Non-telephone telecommunication services**
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2020