QALL-ME
Question Answering Learning technologies in a multiLingual and Multimodal Environment

FP6 IST-033860

http://qallme.itc.it

Final QALL-ME Architecture

Author: Bogdan Sacaleanu, Günter Neumann

Affiliation: DFKI

Keywords: QALL-ME architecture, cross-language, structured data, semi-structured data, unstructured data

Abstract: This deliverable describes the final architecture of the QALL-ME multilingual and multimodal question answering (QA) system. It builds on the previous intermediary architecture deliverables (QALL-ME_D2.1_20070505, QALL-ME_D2.2_20080215) that outlined a solution for domain-specific question answering based on structured data and extends it to cover semi-structured and unstructured data, open-domain questions, and caching of information in a so-called episodic memory.

Date: 16/02/2009

Document Number: QALL-ME_D2.3_20090216

Status/Version: Final Version

Distribution Level: Public

Project Reference	FP6 IST-033860
Project Acronym	QALL-ME
Project Full Title	Question Answering Learning technologies in a multiLingual and Multimodal Environment
Distribution Level	Public
Contractual Date of Delivery	February, 2009
Actual Date of Delivery	February, 2009
Document Number	QALL-ME_D2.3_20090216
Type	report
Status & Version	Final Version
Number of Pages	
WP Contributing to the Deliverable	WP2
WP Task responsible	DFKI
Authors	Bogdan Sacaleanu, Günter Neumann
Other Contributors	
Reviewer	
EC Project Officer	Xavier Gros
Keywords: QALL-ME architecture, cross-language, structured data, semi-structured data, unstructured data	
Abstract: This deliverable describes the final architecture of the QALL-ME multilingual and multimodal question answering (QA) system. It builds on the previous intermediary architecture deliverables (QALL-ME_D2.1_20070505, QALL-ME_D2.2_20080215) that outlined a solution for domain-specific question answering based on structured data and extends it to cover: ▪ semi-structured and unstructured data, ▪ open-domain questions, ▪ and caching of information in a so-called episodic memory.	

Summary

1. Introduction

This deliverable describes the final architecture of the QALL-ME multilingual and multimodal question answering (QA) system. It builds on the previous intermediary architecture deliverables (QALL-ME_D2.1_20070505, QALL-ME_D2.2_20080215) that outlined a solution for domain-specific question answering based on structured data and extends it to cover:

- semi-structured and unstructured data,
- open-domain questions,
- and caching of information in a so-called episodic memory.

Domain-specific question answering shows good performance when coupled with high accuracy data stored in structured repositories (databases). However, the availability of these data is limited and restricted either by copyright or by high costs of acquisition, making it partially suitable for large-scale question answering systems.

Generally, answers in domain-specific applications are mined from different types of sources, such as web portals and newspapers, covering the given domain (tourism in our case). The information contained in such heterogeneous sources usually appears as a mix of free natural language and HTML semi-structured data, which have to be combined. Semi-structured information often concerns coarse-grained information regarding, for example, locations, addresses, opening/closing times, basic facilities, etc. However, additional (fine-grained) information concerning, for instance, sport events, possibilities for recreation or site-seeing events, is usually described in free unstructured text.

When dealing with domain-specific QA applications it is difficult for a user to draw a clear line between information pertaining to the domain and information out-of-domain. What for the systems could be out-of-domain questions, might be still domain-related questions from a user’s point of view. This issue arises mainly from the difference in domain-knowledge as perceived by the user and as it is represented in the QA system. Since it is a real endeavour to provide an exhaustive representation of domain knowledge to a QA system, we have decided to tackle this problem by realising an extension of the system that deals with open-domain, and implicitly with out-of-domain, questions. Questions considered out-of-domain by the domain-specific QA system are considered for answering by the open-domain QA system that uses unstructured data from the Web.

Domain-specific QA applications, in contrast to open-domain ones, build upon limited knowledge-bases and vocabularies. When dealing with large numbers of users, the chances of two users asking the same or a similar question become high enough to consider keeping some information in memory for quick access the next time it is needed. This process, called caching, makes sense only when the time of accessing the information in memory is considerable shorter than the time of producing the same information all over again. Since the process of answering a question is divided in several intermediary steps, of which some might be identical over different questions, saving the result of some time-intensive tasks in a so-called episodic memory might improve the performance of the system in terms of reaction time.

Section 2 will give an overview of the overall architecture of the system and introduces key abstractions that are independent of the type of data used to extract the answers from. Section 3 describes the the structure and behavior of architecturally significant portions of the textual question answering core and presents two instantiations for structured and unstructured data types. Section 4 introduces the episodic memory caching mechanism.

2. Bird's-Eye View of the Architecture

QALL-ME aims at developing a shared infrastructure for multilingual and multimodal question answering (QA), which includes the basic components that are required for providing the following capabilities:

- Automatically gathering, storing, and updating relevant information extracted from different (structured, semi-structured and non-structured) source data types;
- Dealing with complex multilingual questions, anchored to a spatial and temporal context;
- Dealing with both textual and spontaneous speech access modalities;
- Presenting users with correct, complete, and concise answers extracted from different multilingual sources;
- Combining different output presentation formats (e.g., texts, maps, images).

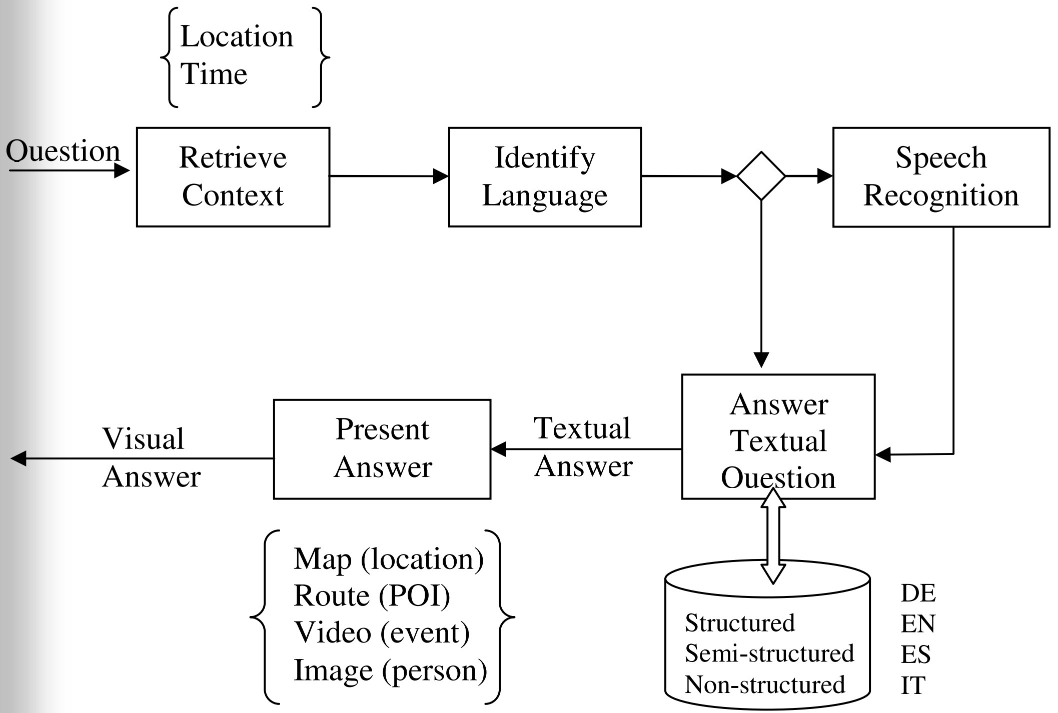

Figure 1. Bird's-Eye View of the QALL-ME Architecture

Error! Reference source not found. presents the general architecture of the QALL-ME system that meets the requirements above mentioned. For a given question *Q* temporal and spatial contexts are determined, based on which relative expressions like *now*, *today*, *tomorrow*, *around here*, etc. can be resolved to the intended meaning of the inquiry. Following, the language of the question is identified and in case of input being in the form of a speech signal it is converted in textual form through use of a *Speech Recognition* module. The core of the system is the *Answer Textual Question* component that provides the main functionality for answering textual question from multilingual source data of different formats: structured, semi-structured and unstructured. The answer is then presented to the user and supported by several additional services: a map in case of answers representing a location, a route for answers referring to a point of interest (POI), a video for answers that are names of events (i.e. movies) and an image for answers referring to a person.

Retrieve Context

Representing and reasoning about time- and location-dependent information is important for applications ranging from databases, planning, natural language processing, and others. Temporal and spatial reasoning is essential in question answering to successfully address questions related to events beside those asking for facts.

The conceptual difference between events and facts is that events occur at a definite point in time, while facts are either extended in time, or don't have a specific time attached to them. An event can be defined as *something that happens within a context*, where context includes *the relationship of an event to time, location, and other events*. Simply put, what makes events so *special* is that by definition they include contextual information.

Contextual information, both temporal and spatial, can be either manually specified by the user or automatically elicited from external sensors like GPS in case of mobile applications. Knowing details about question's context, both in terms of time and location, helps the QA system in better specifying the meaning of the inquiry and therefore delivering better answers.

Identify Language

In a multilingual scenario the input questions can be in any of the languages supported by the system. Correctly identifying the language of the question is a crucial point for downstream components (Speech Recognition and Answer Textual Question) that are language-dependent and whose performance is bound to a correct input.

Speech Recognition

The QALL-ME systems allows input in both textual and vocal form. Since the core QA functionality (*Answer Textual Question*) accepts only textual input, a component for converting spoken words into machine-readable input is demanded. The Speech Recognition module first digitizes the speech signal into a set of useful features that are then used to search for the most likely word sequence, making use of constraints imposed by domain-specific acoustic, lexical and language models (see QALL-ME_D6.2_20081115).

Answer Textual Question

The core component of the QALL-ME system providing question answering functionality is the *Answer Textual Question*. It accepts as input a question in textual form along its context and extracts answers from several types of data (structured, semi-structured and unstructured) in different languages (DE, EN, ES and IT). The system combines a domain-specific approach to question answering based on textual entailment with an open domain approach based on the assumption of redundancy being a good indicator of answer suitability.

Present Answer

Depending on the type of the answer found additional supporting evidence is presented to the user beside the textual result of the QA core. For answers representing a geographical location the system provides a map of its surroundings, on which several domain-specific points of interest are visually marked. For example, in the tourism domain facilities like cinemas, hotels, restaurants, etc. are pinpointed and location of the user is marked, when available. For answers related to a point of interest (i.e. cinema) a map indicating the route to it along textual information about distances and directions is provided. When the answer is an event (i.e. movie) video information about it (movie trailer) is retrieved from online free-available services like YouTube and in case of a person being referred images are presented from through services like Google Image Search.

3. Textual Question Answering Architecture

Question Answering systems cover a wide scope of different methods and architectures, such as question type ontology, external databases of world knowledge, heuristics for extracting answers of certain types, reasoning through inference rules, feedback loops, generation of answers, machine translation, machine learning and even logical analysis, so that it is nearly impossible to capture all within a single architecture.

The systems developed up-to-now, though, share a common pipeline architecture (*Figure 2*), combining three essential modules in a sequential manner: *question analysis*, *information retrieval*, and *answer extraction and selection*.

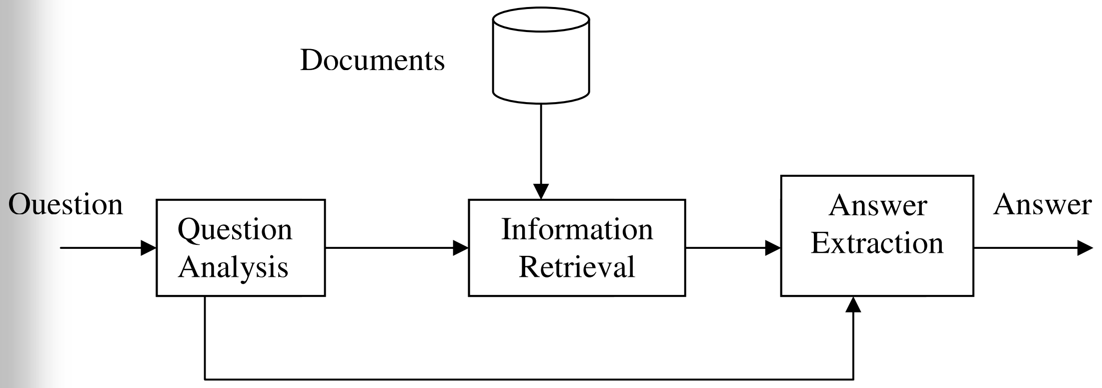

Figure 2. A generic QA System architecture.

The *Question Analysis* module processes the questions (e.g. part-of-speech tagging, named entity extraction, parsing), analyzes and classifies them according to different ontologies. At this stage information related to the question semantic and expected answer type is extracted, which triggers different strategies of retrieval and answer extraction later on.

The *Information Retrieval* module generates queries according to question types, keywords, and additional content. Based on these queries, relevant documents expected to contain correct answers are retrieved.

The *Answer Extraction* module extracts candidate answers from relevant documents and assigns them a confidence score, following to select the most probable answer as correct based on notions of overlap and similarity.

The QALL-ME core functionality for question answering has been designed in line with the generic QA architecture and covers both domain-specific and open-domain scenarios, as well as different types of data in terms of structure. ***Figure 3*** outlines the common core QA architecture of the QALL-ME system with major sub-components for each of the major modules *Question Analysis*, *Information Retrieval* and *Answer Extraction*. At this level of detail the architecture covers the key abstractions for both structured and unstructured data, for both domain-specific and open domain scenarios of use.

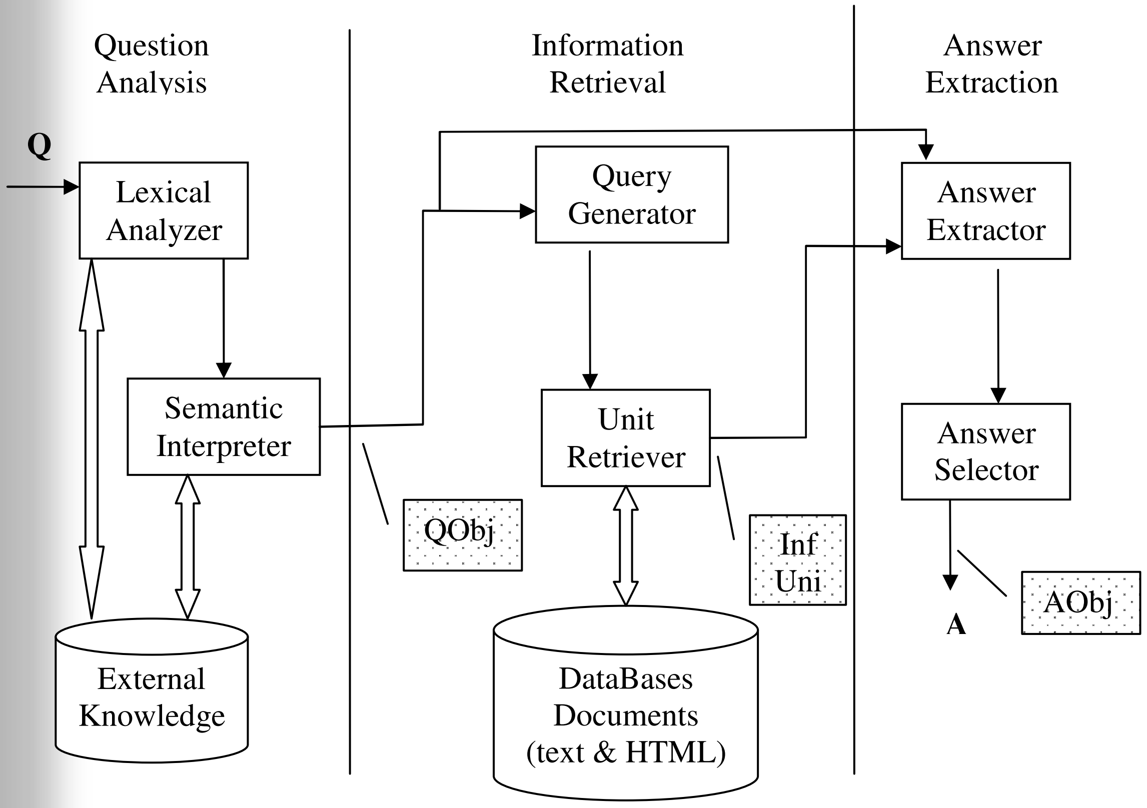

Figure 3. Core QA Architecture for QALL-ME.

Question Analysis

The *Question Analysis* module builds on the assumption that the natural languages covered by the system (i.e. English, German, Italian, Spanish) obey the principle of compositionality, according to which the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them. The principle of compositionality states that in a meaningful sentence, if the lexical parts are taken out of the sentence, what remains will be the rules of composition.

Therefore, the *Question Analysis* module consists of two sub-components: the *Lexical Analyzer*, whose role is to identify the lexical parts (terms) of interest for the given domain, and the *Semantic Interpreter* that discovers the relations between the terms in order to build a representation of question's meaning (*QObj*). During the process of constructing a semantic representation of the question both sub-components make use of external knowledge sources, either domain specific or not, to access the meaning of the lexical parts (*Lexical Analyzer*) or to identify the correct relation between terms (*Semantic Interpreter*).

Information Retrieval

Information Retrieval is concerned with retrieving from a large data collection those parts that are in some way relevant to a given query. The *Information Retrieval* module takes care of preparing the data for quick access (indexing) and transforming the information request (*QObj*) to a query appropriate for the search engine (*Unit*

Retriever). Since the data can be in different formats (database entries, text, HTML) and supported by various search technologies, a way of translating the information need into an specific query language (SQL, SPARQL, etc.) is needed. The *Query Generator* receives a QObj and generates a query for every type of search engine.

The result of the retrieval process consists in a set of question-relevant ranked information units (*InfUnit*) containing possible answers, their context and a similarity score for each. The *InfUnit* is an invariable encapsulation of the relevant information, independent of the original data type or the process that generated it and therefore a stable representation for the modules downstream.

Answer Extraction

The *Answer Extraction* component of a question answering system is one of the most critical but also one of the most difficult stage in the process of finding exact correct answers to questions. Given a piece of text (*InfUnit*), an answer extractor identifies candidate answers and makes a decision whether each candidate is a correct answer or not. Most question answering systems' answer extractors compute scores based on their content and structure, as well as on the content and structure of the corresponding textual contexts.

The *Answer Extractor* sub-component extracts all answer candidates based on contraints imposed by the question through the *QObj*. Only those candidates over a specific threshold defined in terms of constraint fullfilment will be considered for further processing and passed over to the *Answer Selector*. The Answer Selector scores the answer candidates according to the semantic similarity of their textual context with the question, whereby the principle of compositionality is applied. That means, that both overlap in lexical parts and way of their composition play a significant role in determining the most appropiate answer to the question.

1.1 Architecture Realization for Structured & Semi-Strucutred Data

Figure 4 depicts the details of implementation for the case when the data collection is of structured and semi-structured types. We consider as structured the data that has an invariable schema over time, all attributes are present and is mainly contained in domain-specific databases. Semi-structured data on the other side has variable schema over time, part of attributes may be missing and is to be founded in Web HTML pages.

The approach used for answering questions from structured and semi-structured data is based on an *interlingua approach*. That is, both the question and the data collection are mapped to an intermediary representation and answering the question resumes at finding the best match for question's representation among those of the document collection. For QALL-ME we have used domain specific ontologies to provide a common representation in terms of concepts and relations. That is, assuming that the ontology represents the given domain, concept instances and relations among them have been extracted from the data collection and answering a question consists in following the same process on the question side and finding the best match based on representation overlap.

Following is a short description of each individual sub-component. For a detailed account of this approach please see QALL-ME_D2.2_20080215.

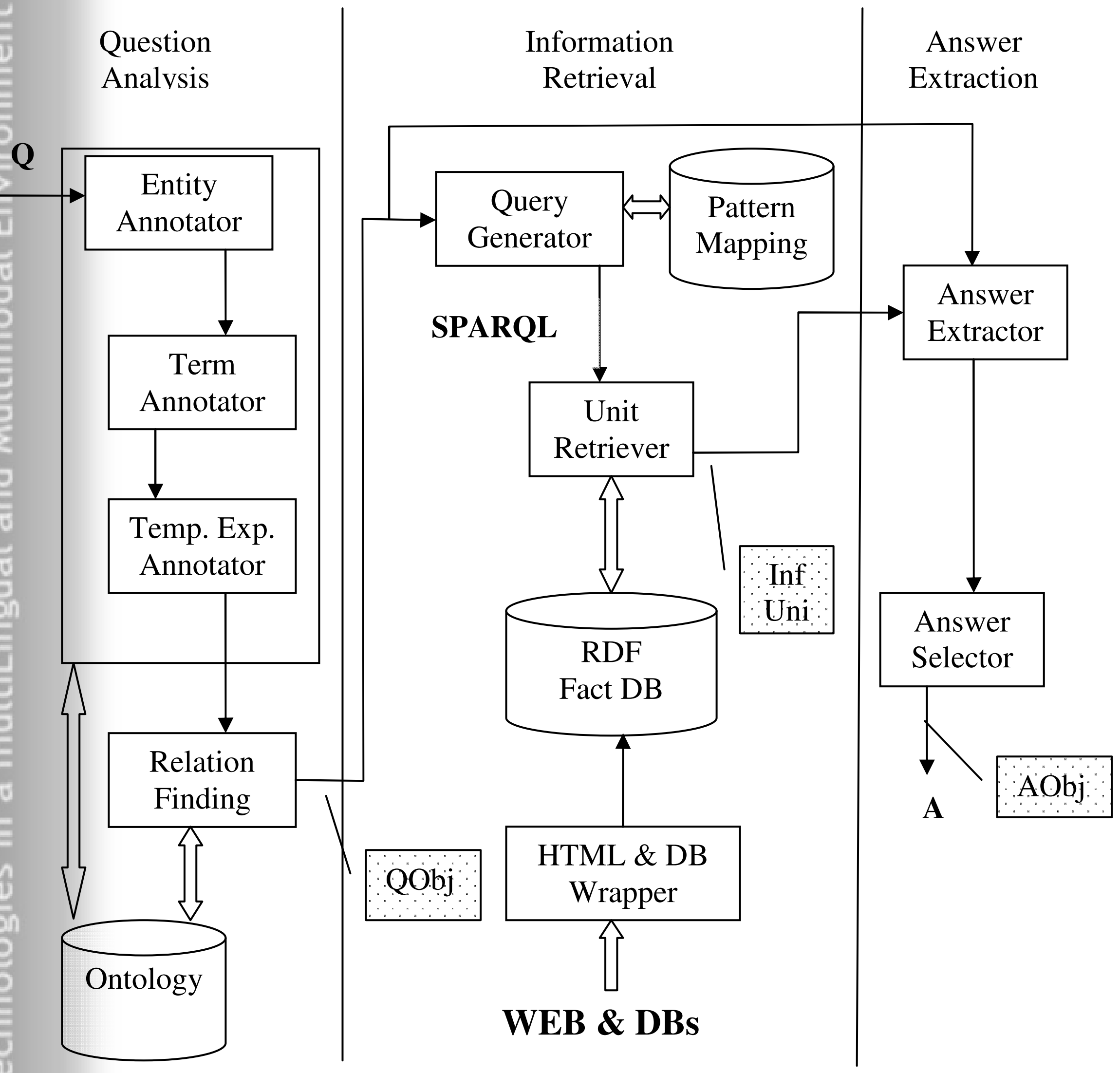

Figure 4. QA Architecture for Strucured & Semi-Strucutred Data.

Question Analysis

The *Entity Annotator* sub-component recognizes named entities that are instances of concepts in the ontology and dependent on the data collection. It will recognize, for example, names of cinemas (*Astra*, *CineStar*), movies (*Dreamgirls*, *300*), cities (*Trento*, *Saarbrücken*), etc. that occur in the data collection and change for every new collection.

The *Term Annotator* marks language specific terms, such as hotel facility concepts (*swimming pool*, *internet access*) that are modelled in the ontology and independent of the data collection.

The *Temporal Expression Annotator* marks simple date and time expressions (*2007-03-15*) and more language specific instances like relative temporal specifications (*today*, *next week*).

The *Relation Finding* sub-component looks for possible relations between the terms extracted so far by using a textual entailment approach. Given a set of lexicalizations for relations from the ontology (i.e. *MOVIE plays in CINEMA, EAType plays in*

CINEMA, MOVIE plays in EAType), it computes the likelihood of the term-marked question (i.e. *<EAType>Where</EAType> does the <MOVIE> Dreamgirls </MOVIE> play?*) of being entailed in one of those relations. Once the most likely relation has been identified and the lexical terms marked, the system has built a semantic interpretation of the question (*QObj*) according to the knowledge covered by the referenced ontology.

Information Retrieval

The first step during this phase is mapping the data to an intermediary representation in terms of concepts and relations from the ontology. This can be done either manually or automatically, depending on the amount of data and its type. For the case of structured data, when schema remains constant over time, a manual mapping method can be chosen, while for semi-structured data that can change its schema over time an automatic approach would be more effective. The *HTML & DB Wrapper* was designed with the goal of automatically mapping the data to the ontology, assuming that both concepts and relations are not ambiguous.

The result of mapping the data collection to the ontology is a repository of factual data expressed in RDF triples format:

MOVIE:Dreamgirls RELATION:plays_in CINEMA:Astra

that represents the information pool for the search module (*Unit Retriever*). It is the role of this search sub-component to retrieve candidate triples (*InfUnit*) that might contain the answer to the information need (*QObj*) expressed through the question.

The role of the Query Generator is to translate the information need (*QObj*) in the query language (SPARQL) of the *Unit Retriever*. Since both represent the information in terms of concepts and relations from the ontology, a repository of mappings between relations and SPARQL queries can be either manually or automatically generated.

Answer Extraction

The expressiveness of the query language used by the *Unit Retriever* might be below that of the natural language and accordingly of the semantic interpretation offered through the *QObj*. Questions asking for the closest restaurant are hard to translate into a query language even though coordinates for points of interest and user's position are given. In such cases the system queries all restaurants in a location determined by user's position along with their coordinates, retrieves a set of InfUnit for candidate answers and postpones the decision of interpreting *closest* to a later time.

It is the goal of the *Answer Extractor* to deal with these cases by computing the distance to all restaurants in the result set from user's position and extract only those meeting the specific constraint. The *Answer Extractor* compensates this way the low expressiveness of the query language by taking over the burden of complex and computational intensive tasks.

Questions asking for the place where a particular movie plays will be answered with the name of some cinemas, for example. Though the answers may be correct, the user expects more detailed information about it, like its address, running hours, etc. The Answer Selector provides the functionality needed to meet this requirement, by selecting more details of the answer to be presented to the user.

1.2 Architecture Realization for Unstructured Data

Figure 5 depicts the details of implementation for the case when the data collection is of unstructured type. By this we mean information found as free-text either on the Web or in a given corpus of documents.

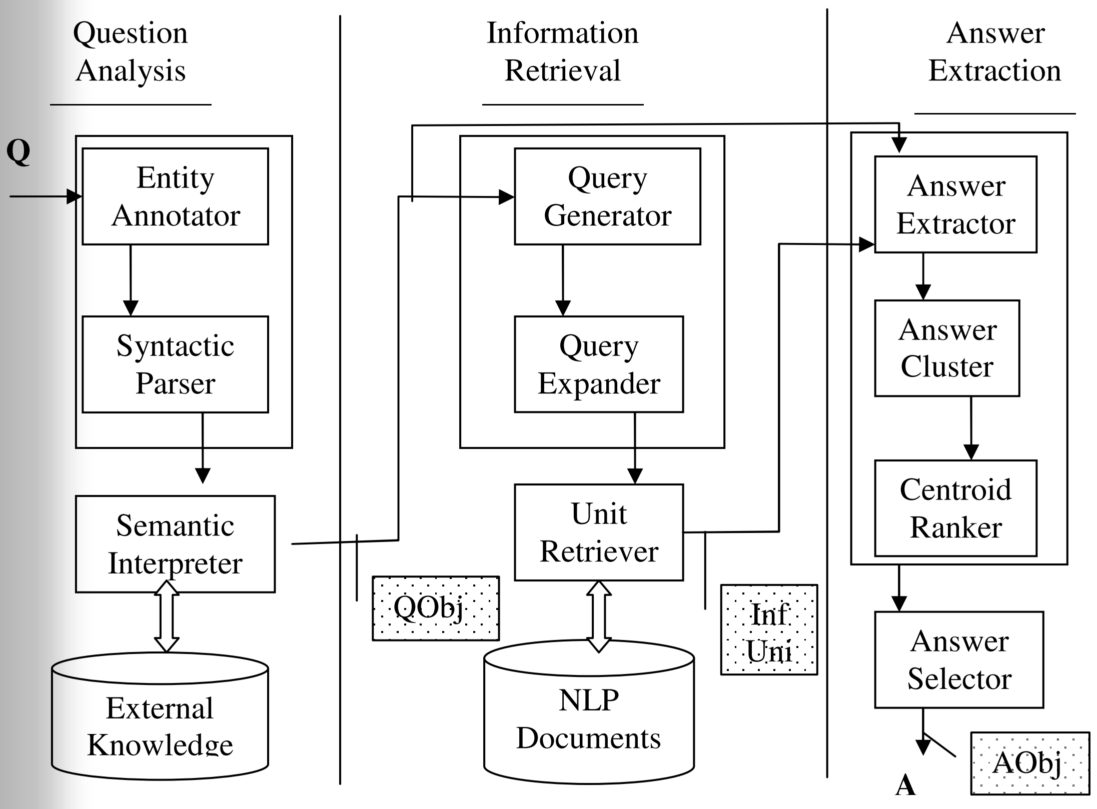

Figure 5. QA Architecture for Unstrucutred Data.

Question Analysis – *What it does?*

The Question Analysis sub-systems reads in the user's information need as a natural-language question (i.e. *How many counties has Austria?*) and generates a formal representation of its meaning, a *QObj*, as presented in Figure 6.

The *question type* (Q-TYPE) is a categorization of questions for purposes of distinguishing between different processing strategies and answer formats. We distinguish between FACTOID and DEFINITION questions with different weighing schemes for their unit retrieval and diverse answer size ranging from word to phrase and even full-length sentence for definitions.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<QOBJ score="1" msg="quest" lang="EN" id="qId0">
  <NL-STRING id="qId0">
      <SOURCE lang="EN" id="qId0">How many provinces does Austria
      have?</SOURCE>
  </NL-STRING>
  <QA-control>
    <Q-FOCUS>provinces</Q-FOCUS>
    <Q-TOPIC>Austria</Q-TOPIC>
    <Q-TYPE restriction="NONE">FACTOID</Q-TYPE>
    <A-TYPE type="atomic">NUMBER</A-TYPE>
  </QA-control>
  <KEYWORDS>
    <KEYWORD type="UNIQUE" id="kw1">
      <TK stem=" province" pos="N">provinces</TK>
    </KEYWORD>
    <KEYWORD type="UNIQUE" id="kw2">
      <TK stem="Austria" pos="N">Austria</TK>
    </KEYWORD>
  </KEYWORDS>
  <NE-LIST>
    <NE type="LOCATION" id="ne0">Austria</NE>
  </NE-LIST>
</QOBJ>
```

Figure 6. Result of Question Analysis.

The expected *answer type* (A-TYPE) represents the class of object sought by the question. Its semantic category drives both the retrieval of segments that contain answer candidates and their actual extraction. We consider the following 8 answer types for FACTOID questions:

- PERSON, e.g.
  Q: Who was called the "Iron-Chancellor"?
  A: Otto von Bismarck.
- TIME, e.g.
  Q: What year was Martin Luther King murdered?
  A: 1968.
- LOCATION, e.g.
  Q: Which town was Wolfgang Amadeus Mozart born in?
  A: Salzburg.
- ORGANIZATION, e.g.
  Q: What party does Tony Blair belong to?
  A: Labor Party.
- MEASURE, e.g.
  Q: How high is Kanchenjunga?
  A: 8598m.
- COUNT, e.g.
  Q: How many people died during the Terror of Pol Pot?
  A: 1 million.

- OBJECT, e.g.
  Q: What does magma consist of?
  A: Molten rock.
- OTHER, i.e. everything that does not fit into the other categories above.
  Q: Which treaty was signed in 1979?
  A: Israel-Egyptian peace treaty.

and the following 4 answer types for DEFINITION questions:

- PERSON, i.e. questions asking for the role/job/important information about someone,
  Q: Who is Robert Altmann?
  A: Film maker.
- ORGANIZATION, i.e. questions asking for the mission/full name/important information about an organization, e.g.
  Q: What is the Knesset?
  A: Parliament of Israel.
- OBJECT, i.e. questions asking for the description/function of objects, e.g.
  Q: What is Atlantis?
  A: Space Shuttle.
- OTHER, i.e. question asking for the description of natural phenomena, technologies, legal procedures etc., e.g.
  Q: What is Eurovision?
  A: Song contest.

Both the question type and the expected answer type are salient information for a good performance of downstream components and failure to correctly determine them will deem the system unusable in most of the cases.

The question *focus* (Q-FOCUS) represents the property or entity that is being sought by the question and may or may not appear in the context of the correct answer being in most of the cases implied by it (e.g., country, city, name, age, date).

The question *topic* (Q-TOPIC) is the object (person, organization, …) or event that the question is about, whose meaning must appear in the context of the right answer.

The *Question Analysis* is also responsible for extracting additional constraints that the correct answer has to satisfy. Such constraints can take different forms like keywords and named entities. The keywords of the question might contain, beside the focus and the topic, lexicalizations of the relation between the two, usually in the form of a verb, and dependents or modifiers of them, which put further constraints on their meaning. Named entities recognition is also an integral part of the *Question Analysis* due to their special treatment during retrieval of relevant information and extraction of candidate answers.

Question Analysis – *How it does it?*

The *Question Analysis* sub-system consists of three components: an *Entity Annotator*, a *Syntactic Parser* and a *Semantic Interpreter*. The *Entity Annotator* is mainly responsible for recognizing named entities and annotating them with their semantic type, according to the classification imposed by the test collection (i.e., PERSON, LOCATION, ORGANIZATION, DATE).

The *Syntactic Parser*'s role is that of providing a list of lexical dependencies that hold between the words of the question, which form the basis for the next component. The *Semantic Interpreter* builds upon these dependencies and a set of hand-crafted lexico-syntactic rules to determine the control information of the *QObj*. In this process it makes use of an external knowledge base of entities that provide hints for the expected answer type based on the focus of the question (e.g. *In which city ….* → focus:*city* → answer type: *LOCATION*).

Unit Retrieval – *What it does?*

We call preemptive annotation the process through which the document collection is marked with information that might be valuable during the retrieval process by increasing the accuracy of the hit list. Since answers tend to co-occur with information from question's context in rather compact textual segments The document collection has been anticipatory annotated with sentence boundaries. Moreover, because the expected answer type for factoid questions is usually a named entity type, annotating the documents with named entities provides for an additional indexation unit that might help to scale down the range of retrieved passages only to those containing the searched answer type. The same practice applies for definition questions given the known fact that some structural linguistic patterns (appositions, abbreviation-extension pairs) are used with explanatory and descriptive purpose. Extracting these kinds of patterns in advance and looking up the definition term among them might return more focused results than those of a search engine based solely on words.

Unit Retrieval – *How it does it?*

The *Query Generator* process mediates between the question analysis result *QObj* (answer type, focus, keywords) and the search engine (factoid questions) or the repository of syntactic structures (definition questions) serving the retrieval component with information units (passages). The *Query Generator* process builds on an abstract description of the processing method for every type of question to accordingly generate the IR query to make use of the advanced indexation units. For example given the question "*What is the capital of Germany?*", since named entities were annotated during the offline annotation and used as indexing units, the Query Generator adapts the IR query so as to restrict the search only to those passages having at least two locations: one as the possible answer (*Berlin*) and the other as the question's keyword (*Germany*), like the following example shows:

+text:capital +text:Germany +neTypes:LOCATION +LOCATION:2.

It is often the case that the question has a semantic similarity with the passages containing the answer, but no lexical overlap. For example, for a question like *Who is the French prime-minister?*, passages containing *prime-minister X of France*, *prime-minister X … the Frenchman* and *the French leader of the government* might be relevant for extracting the right answer. The Query Extension component accounts for bridging this gap at the lexical level, either through look-up of hand-crafted unambiguous resources (e.g. *French ~ France ~ Frenchman*) or searching external resources like wordnets and thesauri for synonyms and conceptually related terms (e.g. *prime-minister ~ government leader*).

In the context of our experiments two different settings have been considered for the retrieval of relevant textual segments for factoid questions: one in which a passage consists of only a sentence as retrieval unit from a document, and a second one with a window of three adjoining sentences for a passage. Concerning the query generation, only keywords with following part-of-speeches have been considered for retrieval: nouns, adjectives, adverbs and verbs, whereby only nouns and adjectives are mandatory to occur in the matching relevant segments. In case of empty hit list retrieval, the query undergoes a relaxation process maintaining only the topic of the question, its modifiers and the expected answer type (as computed by the Question Analysis sub-system) as mandatory items:

Question:	How many provinces does Austria have?
IR-Query:	+neTypes:LOCATION +text:province +text:Austria^4 text:have
Relaxed IR-Query:	+neTypes:LOCATION text:province +text:Austria^4 text:have

Answer Extraction - *What it does?*

The *Answer Extraction* sub-system is based on the assumption that the *redundancy* of information is a good indicator for its suitability. The different configurations of this component for factoid and definition questions reflect the distinction of the answers being extracted for these two question types: simple chunks (i.e. named entities and basic noun phrases) and complex structures (from phrases through sentences) and their normalization. Using the most representative sample (centroid) of the answer candidates' best weighed clusters, the Answer Selector sorts out a list of top answers based on a *proximity* metric defined over a graph representation of the answer's context.

Answer Extraction – *How it does it?*

Based on the control information supplied by the *Question Analysis* sub-system (Q-TYPE), different extraction strategies are being triggered (noun phrases, named entities, definitions) and even refined according to the A-TYPE (definition as sentence in case of an OBJECT, definition as complex noun phrase in case of a PERSON).

Whereas the *Answer Extractor* process for definition questions is straightforward for cases in which the offline annotation repository lookup was successful, in other cases it implies an online extraction of those passage-units only that might bear a resemblance to a definition. The extraction of these passages is attained by matching them against a lexico-syntactic pattern of the form:

<Searched Concept> <definition verb> .+

whereby <definition verb> is being defined as a closed list of verbs like *is*, *means*, *signify*, *stand for* and so on.

For factoid questions having named entities or simple noun phrases as expected answer type the Answer Clusterer (normalization) process consists in resolving cases of co-reference, while for definition questions with complex phrases and sentences as possible answers more advanced methods are being involved. The current procedure for clustering definitions consists in finding out the focus of the explanatory sentence or the head of the considered phrase. Each cluster gets a weight assigned based solely on its size (definition questions) or using additional information like the average of the IR scores and the document distribution for each of its members (factoid questions).

Within the *Answer Selector* the context is first normalized by removing all functional words and then represented as a graph structure. The score of an answer is defined in terms of its distance to the question concepts occurring in its context and the distance among these.

4. Episodic Memory

The Question Answering process can be seen as a factorization of three sub-processes: analyses of question, retrieval of relevant documents and extraction of answers. Each of them are stand-alone units with clearly defined interfaces in terms of input/output objects: *QObj*, *InfUnit* and *AObj*. Answering a specific question implies calling all three units in a pipeline, with output of one process being passed as input to the next one down the stream. Failure of one unit to process the data and pass a result to the next unit in the pipeline will let the question unanswered. In order to avoid this kind of issues, the QA system could try to resend the question again through the pipeline, from the beginning, when the unit that failed recovers. This would implicitly mean calling again all the units, even though in a previous run some of them already processed the question. Since some of the components might take considerable time compared to others, a good idea is to save intermediary results for each question in a separate repository, till the system has completed the pipeline and the answer is delivered.

Beside the scenario above-mentioned when saving intermediary results can improve the response time of the system in case of failure for one of the components, a caching mechanism can also be beneficial when the system is working properly, but some components take long time to process the data. Keeping the information in the episodic memory can spare the working time allocated to a process, if the data being processed has already been seen. For example, if two different users pose the same question within a specific time limit, which is the life of information in the memory, then the system can directly return the answer from the cache to the second user without the need of passing over again the question through all the processes down the line.

In the Question Answering scenario we distinguish between several cases in saving information in the episodic memory for quicker access. For the clarity of explanation we will consider a question representing an incomplete binary relation R(X,Y), where one of it's attributes is missing and corresponds to the expected answer: R(X,?). For example, given the relation PLAYS_IN(MOVIE, CINEMA) and the question *Where does Dreamgirls play?*, the semantic representation can be abstracted to PLAYS_IN(Dreamgirls,?). The workflow of the Question Answering system could be outlined as it follows:

Question:	Where does Dreamgirls play?
Lexical Analysis:	Where does <MOVIE>Dreamgirls</MOVIE> play?
Semantic:	PLAYS_IN(Dreamgirls, ?CINEMA?)
Query:	select CINEMA from DATA where MOVIE=”Dreamgirls” and REL=”plays_in”

Answer: ASTRA

The several cases of caching are related to the kind of information that is saved in memory:

- Question – Answer
- Lexical Analysis – Query
- Semantic representation – Answer

and correspond to the following scenarios:

- The question has been already asked before.
    - *Q*: Where does Dreamgirls play?
    - *A*: ASTRA
- The relation representing the question has been seen before, but with other terms (lexical units).
    - *Q*: Where does Casino Royale play?
    - *Lex. Analysis*: Where does <MOVIE>Casino Royale</MOVIE> play?
    - *Query*: select CINEMA from DATA where MOVIE=” Casino Royale” and REL=”plays_in”
- The relation representing the question has been seen before, but with other lexical representations.
    - *Q*: Where can I see Dreamgirls?
    - *Semantic*: PLAYS_IN(Dreamgirls, ?CINEMA?)
    - *A*: ASTRA

An important aspect of caching data about events is the spatial and temporal context of the question. The system can face the same question coming from two users in different locations. Failing to capture the context of a question like *Where can I see Dreamgirls today?* might result in retrieving results of a previous identical question that has been saved for another location and another day. Therefore the episodic memory must maintain a separate register for a question’s context, beside those for the *Question Analysis*’s result.