UMagic! THE UML Modeler for Text Documents

Iram Shahzadi, Qanita Ahmad, Kiran Fatima, Imran Sarwar, and Waqar Mahmood

***Abstract*— In this paper we present an open source text mining system, UMagic, which identifies key domain entities and their relationships from text documents. UMagic extracts the key information components from the textual document and transforms them into UML based diagrams without human intervention. UMagic performs the linguists processing of given text using open source tool named GATE [1], to mark entities and relationships between these entities. Afterwards, it generates ER diagram from the marked text automatically. Though the task in hand is very complex, specifically when carried out in absolutely automated fashion, but it has immense applications in real world scenarios. From Software Engineering perspective, this approach can be employed to bridge the gap between the analysis phase and design phase of the software development process. This results in reduced time and complexity of the design phase, as well the improved degree of correctness of the design documents.**

***Index Terms*—Artificial intelligence, ERD, GATE, natural languare processing, text mining, UML, XML.**

I. Introduction and Motivation

Today, we live in a world with immensely large amounts of data and information. According to literature, most of the available data is in unstructured form, mainly containing textual information [2], [3]. It is not easy to analyze this unstructured textual data to extract meaningful knowledge instances from it. For this purpose, research in text mining is gaining importance for processing textual information.

There exist two main phases for text mining: a preprocessing phase which converts the text into a structured or semi-structured form, and secondly discovery phase in which text mining techniques are applied to extract the information nuggets. The purpose of this paper is to use the concept of text mining to draw UML diagrams from textual documents. Since the manual design of such diagrams requires time and expertise, we present a modeling system UMagic, The UML Modeler. UMagic take textual information as input, processes it, converts it into a formal data-model, and generates structured diagrams for the system. Thus, the system is deemed to simplify the design process in software development and business field.

Entity Relationship Diagram (ERD) is the most commonly used illustration of the relational data-model for any software or business system [4]. It provides the foundation for relational database architecture [5]. In other words, ERD mocks up the logical model of a relational database. The constituents of an ERD are entities, their attributes, and their inter-relationships. Generally, entities are considered as nouns and relations as verbs. Three major types of relations that connect entities in a database are: One-To-Many (1: m), One-To-One (1:1) and Many-To-Many (m: m).

To process textual information, for the ERD generation, is quite hectic especially for complex systems. We can use some open source linguistic processing software like General Architecture for Text Engineering (GATE) for initial textual processing and rendering the information into structured format. For the purpose of annotation, GATE employs Language Resources (LRs) and Processing Resources (PRs). LRs are used to create the corpus, on which PRs works to perform different operations required to process underlying text. GATE Visual Studio provides the functionality to draw diagrams. The resulted diagram is in the pictorial form showing precise and correct information. The information about the drawn diagram is stored in the XML format.

The rest of the paper is organized as: section 2 described the related work and background study of the relevant concepts. Section 3 illustrates in detail the framework for UMagic, section 4 presents a discussion of the outcomes of the proposed system. Finally, we conclude the paper and give future directions in section 5.

II. Related Work and Background

Text mining has gained much popularity and its usage is increasing day by day. Text mining applications facilitate users by automatically extracting the information of interest. During different phases of software development lifecycle, various documents i.e. requirement analysis document, functional specification document; design document etc. are generated. Employing text mining applications to process these documents may help in extracting useful pieces of information. For example if requirement analysis document is processed by applying text mining techniques, it can be transformed into a formal data-model, and structured diagrams can be drawn for the system in automated fashion. This can also help further in automatic generation of database for a software system, reducing the workload of software engineers.

Today, relational database is the most appropriate way to store the information digitally in an organized manner. Relational database provides the facility to manipulate this stored information according to the underlying requirements. Design of relational database revolves around the concept of entities and relationships between them. In common words, an entity is a real world thing that has some properties called attributes. An entity is composed of a set of attributes that capture the information about that entity. Key attribute or a set of key attributes are used to relate one or more entities.

Manuscript received June 25, 2012; revised September 8, 2012.

Iram Shahzadi, Qanita Ahmad, Imran Sarwar and Waqar Mahmood are with Al-Khawarizmi Institute of Computer Science University of Engineering & Technology, Lahore, Pakistan (e-mail: iram.naseer@kics.edu.pk)

Kiran Fatima is with Department of Computer Science & Engineering, University of Engineering & Technology, Lahore, Pakistan (kiran_csengineer@yahoo.com)

So a relation is a key property of an entity that creates its mapping with different entities [6], [7].

A number of tools exist in literature that transforms the given ERD into relational database. An XML-based ER-diagram Drawing and Translation Tool, named "ERDraw" presented by Shuyun Xu and others [8], provides a visual interface to draw ER diagram by dragging and dropping the provided ER-diagrams components. ERDraw further works on the user drawn ER diagram, to extract entities and their relations. It generates an ER semantic object model, based on the extracted entities and relations. Semantic object model is basically a data structure capturing the complete ERD information. This Semantic object model is used to generate a relational object model that is transformed into a relational database schema. TabletERD [9] is another tool that provide user with the facility to draw ERD. User can visually draw entities and can drop a relationship between them. TabletERD generates xml and SQL code based on ERD which is compatible with different famous database management systems like MySQL, IBM DB2, Microsoft SQL Server and Oracle.

As discussed above, the existing tools require input from user in the form of entities and their relationships. Whereas our proposed tool UMagic automatically performs the task of extracting entities from the given text, along with their relations. Performing this task in automated manner, results in faster drawing of ER-diagram and further generation of entity relational data model. Changes are easy to incorporate as most of the work is to be done by the system not by the user.

III. Our Proposed Methodology

UMagic is the blend of text engineering with UML. It takes text document as input and processes it to add annotations with this text. From Software Engineering perspective UMagic can be employed to bridge the gap between the analysis phase and design phase of the software development process. This results in reduced time and complexity of the design phase, as well the improved degree of correctness of the design documents.

UMagic, the UML modeler proposes the idea to annotate the given text automatically using GATE language processing tool and extract entities and their relations from this annotated text. The extracted information, in terms of entities and their inter-relationships, can be used to draw ER diagram. The entity relation diagram serves as the basis to the physical data model for the system, and thus is of the outmost importance in the software development lifecycle.

We employ the linguistic processing capabilities of GATE to extract the entities and relationships information from the input textual data. For the purpose of basic text processing and information extraction, GATE provides ANNIE, A Nearly New Information Extraction. ANNIE consists of different components such as tokenizer, gazetteer, sentence splitter, semantic tagger, and part-of-speech tagger processing of the underlying documents. For the implementation purposes of UMagic, we used ANNIE to parse different parts of speech from the textual data. The nouns are marked as entities; while verbs as relations. This tagged information is then passed onto ERD generator module for the creation of entity relation diagram.

The system architecture of UMagic is depicted in Fig. 1. The architecture is showing full functionality of the system. Document Acquisition, Document Processing, XML Modeling and ERD Generation are the main modules of the system. Document Acquisition obtains document for processing. Then Processing is done in Document Processing module through GATE. This processed information of document is stored in the form of XML format. ERD Generation module takes this information as input and draws the required diagram which is ERD in our case. Detailed description of these modules is as follows:

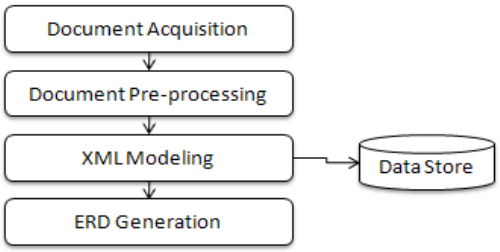

Fig. 1. UMagic system architecture

A. Document Acquisition

This module is responsible for the collection of textual document to provide input to the system. It invokes the relevant interface in the document processing module for the further linguistic processing on the document. The document acquisition module takes the input textual documents in a variety of formats like pdf, html, plain text files, etc.

B. Document Processing

Document processing module serves as the backbone of the system. It uses the plug-ins provided by GATE to process the document linguistically. As mentioned earlier, ANNIE, the information extraction system of GATE is employed to mark the required information. The core components required for this module are discussed here.

1) Sentence splitter

The main functionality of sentence splitter is to segment the input textual document into sentences. The splitter uses a gazetteer list of abbreviations to help distinguish the end of a sentence. The sentence splitter is domain and application-independent. The most commons forms of sentence splitter abbreviations are '.', line breaks, '?', multiple punctuations '??!!?' etc.

2) Tokenizer

The tokenizer breaks up the text into 'tokens' which normally are strings of characters, categorized according to the defined rules as a symbol. The tokenizer splits the text into very simple tokens such as numbers, punctuation and words of different types. This step is essential for the further morphological analysis and information extraction from the input text. No further processing on the text is possible in ANNIE without this step.

3) Part-of-Speech tagger

This component annotates the tokenized text. Each word or symbol is tagged with its part of speech i.e. verb noun, adjective, conjunction etc. ANNIE uses a given lexicon and

a set of rules for the purpose of annotations.

4) Semantic tagger

Semantic tagger works on the already tagged pieces of text and transforms them into annotated entities by using the Java Annotation Patterns Engine (JAPE) provided by the GATE. JAPE uses rules defined in JAPE language for annotating these entities.

```
Rule: Offical_fName
(
        {Lookup.majorType == fName }
): Offical
-->
:Official .fName = {rule = "Official_fName ", type
= :Official.Lookup.minorType}
```

Fig. 2. Gazetteer rule

5) Gazetteer

Gazetteer lists are plain text files containing one entity name per line. The purpose of gazetteer is to identify the entities, by string matching, in the given text and mark them with the hierarchy that is already defined by the gazetteer lists.

6) Orthographic Coreference

The Orthomatcher module, also known as OrthoMatcher, works on the text proceeds by semantic tagger. Till semantic tagger entities are marked. Now Orthomatcher identify relationships between named entities, along with other rules defined in JAPE. Thus in this step co-reference is added between the entities found by semantic tagger in previous step.

7) Modeling Entities and Relationship into XML

After document processing is completed using ANNIE, processed text along with the marked entities and relations is transformed into XML file, and then stored in an xml data store. The format of this XML file is shown in Fig.3.

IV. Results and Discussion

We used the following input text and applied the above discussed steps in sequence. Required components of ANNIE were added from the GATE which processed the text accordingly. Entities were marked with their relative attributes in the given text, as shown in Fig.4.

```
<? xml version= “1.0” encoding= “utf-8”?>
<ERD>
  <Entities>
    <Entity ID= “A.1” Name= “Official”>
      <Attributes>
        <Attribute ID= “A.1.1” Id= “Id_No.” />
        <Attribute ID= “A.1.2” Name= “FName” />
        <Attribute ID= “A.1.3” Name= “LName” />
        <Attribute ID= “A.1.4” Name= “Rank” />
        <Attribute ID= “A.1.5” Name= “Event_Assigned” />
      </Attributes>
    </Entity>
    <Entity ID= “A.3” Name= “Event”>
      <Attributes>
        <Attribute ID= “A.3.1” Name= “Planned_date” />
        <Attribute ID= “A.3.2” Name= “Event_Duration” />
        <Attribute ID= “A.3.4” Name= “No._of_officials” />
      </Attributes>
    </Entity>
    <Entity ID= “A.4” Name= “Sport complex”>
      <Attributes>
        <Attribute ID= “A.4.1” Name= “Location” />
        <Attribute ID= “A.4.2” Name= “Chief_organizing_individual”/>
        <Attribute ID= “A.4.3” Name= “Total_occupied_area” />
      </Attributes>
    </Entity>
    <Entity ID= “A.5” Name= “Roster”>
      <Attributes>……..
        <Attribute ID= “A.5.1” Name= “Id_no.”>
        <Attribute ID= “A.5.2” Name= “FName”>
        <Attribute ID= “A.5.5” Name= “Event_Assigned”>
        <Attribute ID= “A.5.6” Name= “No._of_Event_Play”>
      </Attributes>
    </Entity>
  </Entities>
  <Relationships>
    <Relationship ID= “R.1” Type= “1-m” E1= “A.2” E2= “A.4”>
    <Relationship ID= “R.2” Type= “1-m” E1= “A.2” E2= “A.3”>
  </Relationships>
  <Instances>
    <Instance ID= “” />
    ….
  </Instances>
</ERD>
```

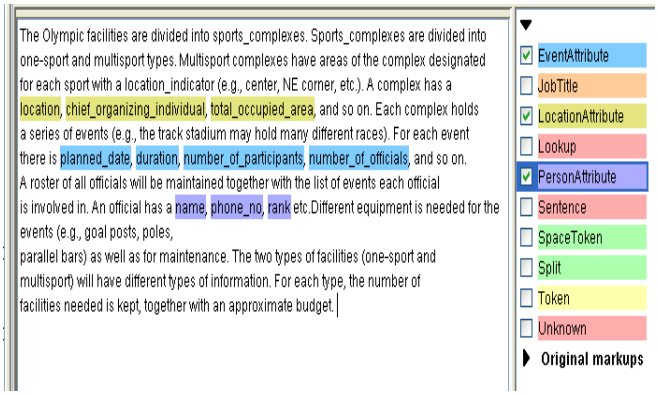

Fig. 3. Xml file format

Following rules were used to mark the entities existing in the given example text.

```
Rule: PersonEntity
 (
({Lookup.majorType == Person}): Person
)
-->: Person.Name = {kind = “Person", rule= PersonEntity}
Rule: LocationEntity
 (
({Lookup.majorType == location}): location
)
-->:location.Name = {kind = "location", rule= LocationEntity}
```

```
Rule: PersonAttribute
 (
{Lookup.majorType == personattributes}
): personattributes
(
{TempPerson}
): person
-->
: personattributes.PersonAttribute = {rule = “PersonAttribute"},:
person.Person = {kind = "personName", rule = “PersonAttribute"}
```

```
Rule: EventEntity
 (
({Lookup.majorType == Event}): Event
)
-->:location.Name = {kind = "Event", rule= EventEntity}
{TempEvent}
): event
-->
: EventAttributes.EventAttribute = {rule = “EventAttributes"},
: event.Event = {kind = "eventName", rule = “EventAttribute"}
): location
-->
: LocationAttributes.LocationAttribute = {rule =
“LocationAttribute"},
: location.Location = {kind = "locationName", rule =
“LocationAttribute"}
```

```
Rule: EventAttribute
 (
{Lookup.majorType == eventattributes}
): eventattributes
(
{TempEvent}
): event
-->
: EventAttributes.EventAttribute = {rule = “EventAttributes"},
: event.Event = {kind = "eventName", rule = “EventAttribute"}
```

Fig. 4. Marked entities' attributes

After the input document is processed, its output is stored in the form of XML, with marked entities and relations; ERD Generation module takes xml file as input and draws the ER diagram as shown in Fig. 5.

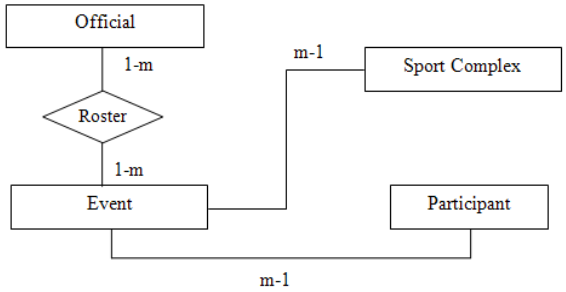

Fig. 5. ER diagram generated as output

As ERD represents the logical data-model on any relational database, it can be easily transformed into physical model i.e into a relational database. Marked entities represented in ERD are transformed into table and attributes of that entity becomes the field of that table. Relations between different entities are used to represent the relations between the tables, representing those entities, as their primary or foreign key. Once database is created, it can be further operated on to find the behavioral model of the generated database attributes in form of their access methods. This can further help in generation of other UML diagrams like Object Diagram, Class Diagram and Interaction Diagrams etc.

V. Conclusion and Future Work

In the paper, we have presented a simple system for the generation of ER Diagrams through the textual document generated during the software development life cycle. UMagic can be helpful in making the project development life shorter. It is user friendly and can be used to store information semantically. The modular approach of the UMagic allows the provision of integration with other tools. For instance, the functionality of UMagic can be extended by providing TabletERD as its plug-in. The TabletERD will ease up the transition of logical model to the physical models through automated generation of database script. The architecture of UMagic is shown to be extensible, in order to cater for the whole array of UML diagrams. As the work presented here processes the text semantically, we intend to work on behavioral aspects of these UML diagrams.

Acknowledgment

We are very thankful to Ayesha Lone, Fabeha Pasha and Umer Saleem, students of Computer Science & Engineering department, University of Engineering & Technology, Lahore, Pakistan. These students helped in practical implementation of this paper.

References

[1] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, I. Roberts, Y. Li, A. Sha_rin, and A. Funk, *Developing Language Processing Components with GATE*, Version 6, User Guide.

[2] J. Rilling, R. Witte, D. K. Gašević, Z. J. Pan, *Semantic Technologies in System Maintenance*, 2008.

[3] M. Penelope, R. Maria, S. Spiros, *Knowledge Mining: A Quantitative Synthesis of Research, Results and Findings*, 2005.

[4] Entity-relationship model. [Online]. Available: http://www.en.wikipedia.org/wiki/Entity relationship_model

[5] Understanding Entity Relationship Diagrams Introduction. [Online]. Available: http://www.folkworm.ceri.memphis.edu/ew/SCHEMA_DOC/comparison/erd.htm

[6] R. Elmasri and S. B. Navathe, *Fundamentals of databse systems* 4th Edition.

[7] C. P. PETER. (1976). The Entity-Relationship Model - Toward a Unified View of Data. [Online]. Available:http://www.en.wikipedia.org/wiki/unified_modeling_language

[8] S. Xu, Y. Li, and S. Lu, ERDraw: An XML-based ER-diagram Drawing and Translation Tool.

[9] S. Sok and C. Scharff, "Work in Progress: Database Design with TabletERD," in *Proc. of Frontiers in Education Conference, 36th Annual*, pp. 20-29, 27-31 2006.