1. **THEORETICAL FRAMEWORK**
   1. **Information Extraction Techniques**
      1. Template-Based Information Extraction Systems

***An Open Architecture for Multi-Domain Information Extraction (Poibeau, 2001)***

Thierry Poibeau has provided a general architecture for developing information extraction systems regardless of its domain (Poibeau, 2001). In his paper, he proposed an information extraction architecture that takes advantage of the capabilities of machine learning to help researchers define new templates (this is where the extracted information is being filled in) with respect to the IE system’s domain.

Poibeau’s architecture is divided into 5 main modules: (1) the module for extracting information from the structure of the text; (2) the module for named entity recognition which is responsible for recognizing places/dates/etc.; (3) the module for the semantic filters; (4) the module for the extraction of specific domain-dependent information; and lastly (5) the module for filling in a result template.

In module 1, a number of information is extracted from the structure of the input text. It is in this module where information that is embedded in the structure of the text is extracted like those that are written in HTML or XML formats. On the other hand, in module 2, relevant information is extracted/recognized through linguistic analysis. This module is responsible for recognizing the different named entities present in the input text like names, places, and dates. Poibeau made use of the finite-state tool *Intex* to develop this module. Furthermore, in module 3, text categorization is performed on the set of so-called “semantic signatures” that were produced from a semantic analysis of the input text. Poibeau made use of the French system Intuition™ to develop this module. In addition, in module 4, specific information like the specific relationships between named entities is extracted by applying a grammar of transducers or extraction patterns on the input text. Lastly, in module 5, all the information extracted from the input text are linked together to fill in a specific result template(s) that present(s) a summarized view of the extracted information. Figure 3.1.1a illustrates the general architecture proposed by Poibeau.

Figure 3.1.1a: Open Architecture for Multi-Domain Information Extraction

* + 1. **Machine Learning-Based Information Extraction Systems**

**TwitIE**

* + 1. **Algebra-Based Information Extraction Systems**

**SystemT**

**An Open Architecture for Multi-Domain Information Extraction (Poibeau, 2001)**

Poibeau, T. An Open Architecture for Multi-Domain Information Extraction. *IAAI-01*. Retrieved May 28, 2014, from www.aaai.org

**SystemT: A System for Declarative Information Extraction (Krishnamurthy et al, 2008)**

Krishnamurthy, R., Li, Y., Raghavan, S., & Reiss, F. SystemT: A system for declarative information extraction. *SIGMOD Record*, *37*. Retrieved May 28, 2014, from http://www.almaden.ibm.com/cs/pro jects/avatar/

Krishnamurthy et al proposed an Information Extraction system that takes advantage of the classical database ideas to overcome the limitations of developing and using grammar-based information extraction systems. The system was called SystemT. In their paper, Krishnamurthy et al provided the architecture that they used in making SystemT. SystemT has two modules or in their case “environments” on which the information extraction is done namely the Development Environment and the Runtime Environment. In the Development Environment, constructing and refining the rules that will be used for the actual extraction is done repeatedly and after which; the rules are registered and specified in a language called the Annotation Query Language (AQL). It is in this environment where the rules are being compiled into algebraic expressions and where the results of executing the rules over a corpus of representative documents are being visualized. Once these results are deemed to be satisfactory, they can now be published into an annotator. The process of publishing the annotator includes the feeding of the AQL into the Optimizer (which compiles the rules into an algebraic expression) and the instantiation of corresponding physical operators by the Runtime Environment. On the other hand, in the Runtime Environment, the environment/module receives a continuous stream of input documents and then it annotates each of the documents and outputs them for further specific processing. Usually, the Runtime Environment is embedded in the processing pipeline of the information extraction system. Figure 3.1.1b illustrates the architecture that was used by SystemT.

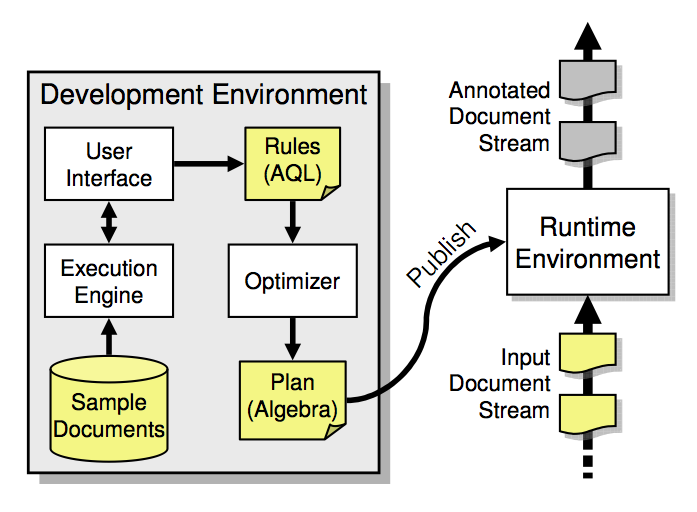


Figure 3.1.1b: The Architecture of SystemT

**TwitIE: An Open-Source Information Extraction Pipeline for Microblog Text (Bontcheva et al, 2013)**

K. Bontcheva, L. Derczynski, A. Funk, M.A. Greenwood, D. Maynard, N. Aswani. TwitIE: An Open-Source Information Extraction Pipeline for Microblog Text. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2013).

Bontcheva and her team of researchers have proposed an information extraction system that is specifically targeted for extracting relevant information from texts coming from microblogs. Bontcheva et al made use of the GATE ANNIE architecture in developing the system and took advantage of some of its built-in tools to further streamline the process of information extraction. In their paper, they presented how they designed the architecture of TwitIE and how they used the existing tools from ANNIE. ANNIE offers a wide array of information extraction tools like tokenizer, sentence splitter, POS tagger, gazetteer lists, finite state transducer (from GATE’s built-in regular expression over annotation language), orthomatcher and coreference resolver but in the case of TwitIE, Bontcheva et al only reused the sentence splitter and name gazetteer because the other components/tools have to be modified to fit the specifics of microblog texts like noisiness, brevity, idiosyncratic language and social context. Overall, TwitIE has the following main components: Language Identifier, Tokenizer, Gazetteer, Sentence Splitter, Normalizer, POS Tagger, and the Named Entity Recognizer. For the Language Identifier, TwitIE made use of the TextCat language identification algorithm, which heavily relies on n-gram frequency models to identify languages. For the Tokenizer, TwitIE followed Ritter’s Tokenization Scheme to treat abbreviations and URL’s as one token and hashtags and mentions as two token. This scheme also features orthography and capitalization preservation. For the Gazetteer, TwitIE used the unmodified version from ANNIE, which compiles lists of entities into finite state machines that can match text tokens. For the Sentence Splitter, TwitIE still used the unmodified version from ANNIE, which is a cascade of finite-state transducers that segments text into sentences. For the Normalizer, TwitIE made use of a combination of a generic spelling-correction dictionary and a social media-specific dictionary. The list of variations is also dynamic by using heuristics to correct spellings. For the POS Tagger, TwitIE made use of the same technique used by a Stanford Tagger, which was trained on tweets that were tagged using the Penn TreeBank Tagset. The improved tagger also included tag labels to support retweets, mentions, URL’s, hashtags and user mentions. Lastly, for the Named Entity Recognizer,. Figure 3.1.1c illustrates the architecture used by TwitIE.

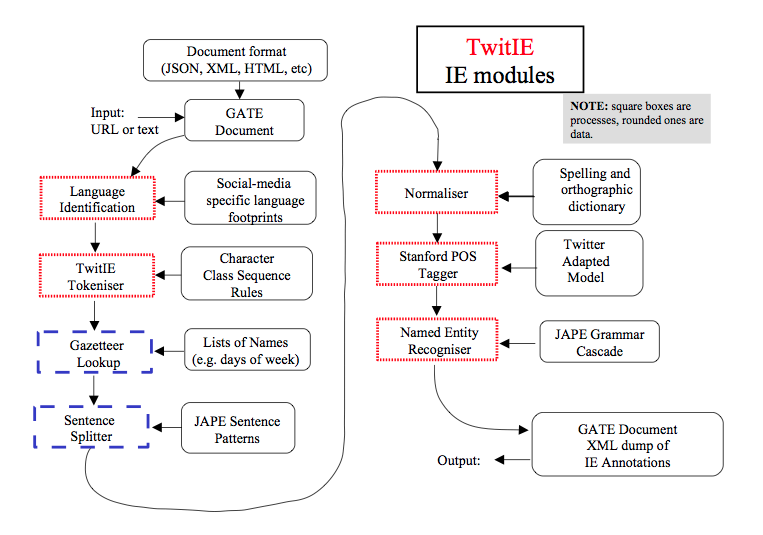


Figure 3.1.1c: The Architecture of TwitIE

**Language Identifier**

The language identification that the system uses is TextCat (Cavnar and Trenkle, 1994), which relies on n-gram frequency models to determine the languages. The TextCat adaptation to Twitter is 97.4% accurate, it currently works on five languages (i.e. English, German, and French).

**Tokenizer**

The system used the Unicode tokenizer bundled with the ANNIE system in GATE. The researchers made some adaptation for special cases found in social media like URLs, hashtags, user mentions, special abbreviations, and emoticons. Given these special elements in microblogs, tokenizing them accurately is important. The tokenizer extracts abbreviation and URLs as one token each, while hashtags and mentions are two tokens (i.e. \# and nike from the hashtag #nike).

**Normalizer**

The TwitIE normalizer consists of a generic spelling-correction dictionary and a spelling-correction dictionary for social media normalizing elements like “2moro” and “brb”. The researchers tried to experiment TwitIE with other normalization tools that resulted to a higher recall (more wrong words can be corrected) but lower precision (some corrections are wrong).

**POS Tagger**

The TwitIE system adapted the Stanford tagger for POS tagging, it is trained on tweets tagged with the Penn TreeBank (PTB) tagset. The tagger was trained using different sets