IsoQuest, Inc.:

Description of the NetOwl™ Extractor System as Used for MUC-7

George R. Krupka   *Kevin Hausman*
IsoQuest Inc.
3900 Jermantown Ave., Suite 400
Fairfax, VA 22030
gkrupka@isoquest.com

INTRODUCTION

IsoQuest used its commercial software product, NetOwl Extractor, for the MUC-7 Named Entity task. The product consists of a high-speed C^{++} engine that analyzes text based on a configuration file containing a pattern rule base and lexicon. IsoQuest used the **NameTag Configuration** to recognize proper names and other key phrases in text, and mapped the product's extraction tags to the MUC-7 NE tags. NetOwl Extractor provides access to the extracted information through a flexible API, and IsoQuest used a small application program to process the documents and write the SGML output.

BACKGROUND

In 1996, SRA International formed IsoQuest, Inc. around its commercial extraction product called NameTag. First released in 1995, NameTag identified proper names and other key phrases and featured a high-speed implementation and ANSI C API. The initial target application for NameTag was processing of news stories to add value for content providers. In MUC-6, NameTag excelled in the Named Entity task with an F-Measure of 96.42, achieving the approximate level of human performance.

Since that time, IsoQuest has moved beyond proper name recognition and news applications. The company has licensed software to over 30 clients for a variety of applications. Many applications require topic categorization or summarization in addition to proper name recognition. Newspaper organizations require the analysis of classified ads in order to publish them online. Financial corporations require software to monitor their email for compliance with regulations. Several companies require the analysis of SEC filings in order to provide immediate access for investors. Some applications require the resolution of names across documents or databases. The variety of applications has pushed IsoQuest to increase speed, decrease process size, add additional APIs, and support multiple threads. These pressures from customer requirements forced IsoQuest to re-engineer NameTag to extend its extraction capabilities while improving its software implementation.

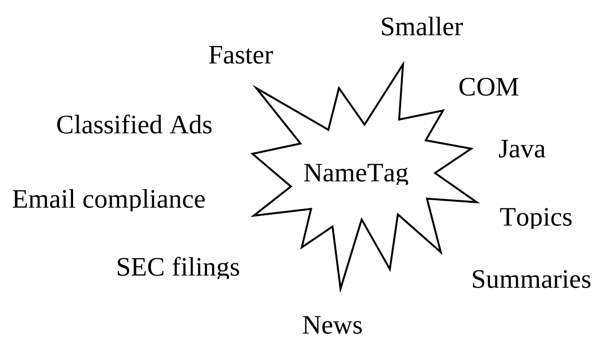

Figure 1: NameTag Under Pressure

SYSTEM DESCRIPTION

IsoQuest's new system is called NetOwl Extractor. The NetOwl Extractor system has been totally redesigned to embody a new extraction architecture. The new architecture clearly separates the run-time processing engine from the declarative configuration data that specifies the type of processing to do. Although previous versions of NetOwl Extractor consisted of a software engine separate from the data configuration files, the software engine contained configuration-specific code and information.

The NetOwl Extractor 3.0 architecture maintains a total separation of the extraction engine and API from the extraction configuration. The engine has no configuration-specific code or information, and the API has a generic framework that applies to any configuration. The specific processing and data that is required to recognize proper names and other key phrases is now contained in a configuration file called the **NameTag Configuration**. However, the same engine and API can perform automotive classified ad extraction based on another separate configuration file, for example.

NetOwl Extractor 3.0 includes a compiler that creates a configuration file from data source files, illustrated in **Figure 2**. At run time, only this configuration file must be loaded, which shortens initialization time and enables efficient memory usage. Compilation is fast so there is no impact on development or customization. (The compilation time was 4.6 seconds for the MUC configuration on a Pentium II 300 Mhz.) Previous versions parsed a significant number of data source files into memory, causing slow initialization and inefficient memory usage. The configuration file includes an extensive set of parameters, allowing more customization of performance during run time.

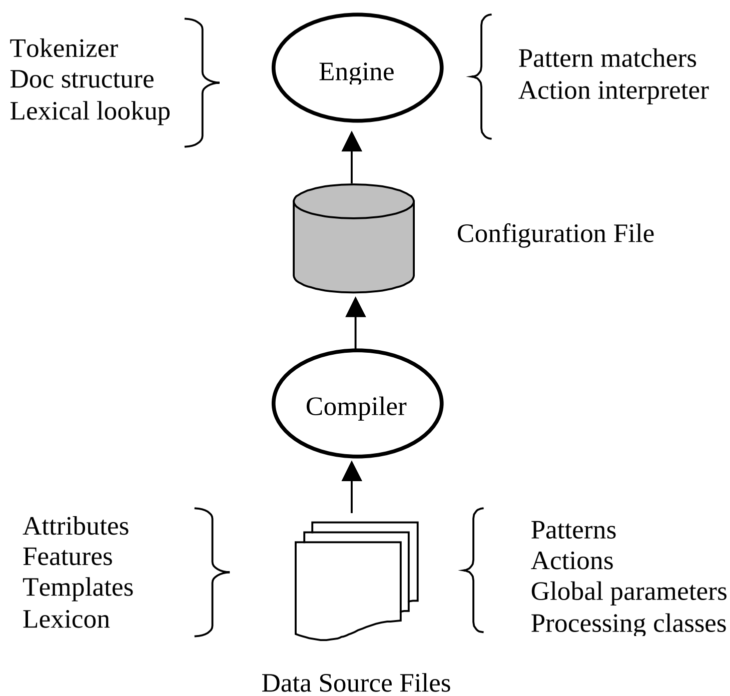

Figure 2: NetOwl Extractor Configuration Compiler

NetOwl Extractor 3.0 now uses generic facilities called *extraction lists* to store the extracted information. Each extraction list stores the results of an extraction processing class, and subsequent classes can rely on the content of other extraction lists. Regardless of the type of extraction, application programs now use one generic method to retrieve the information.

Multiple Configurations

NetOwl Extractor permits a single CPU process to open multiple configuration files. By having multiple configurations open, a single process can perform different types of extraction without having to shutdown and re-initialize the system. For example, a program using NetOwl Extractor could load three configurations, one for news stories, one for SEC filings, and one for email. As it processes documents, it could use the appropriate configuration for each type of document.

NetOwl Extractor also permits processing using more than one CPU thread on those systems that support in-process threads. Given that multiple-CPU machines are becoming more common, this ability lets the user take advantage of several CPUs while only having to run one process. Each thread of processing in NetOwl Extractor may have its own configuration, or several threads may share a single configuration, thus lowering memory usage.

Figure 3 illustrates multiple-configuration (with a single thread), multiple-thread (with a single configuration), and multiple-thread multiple-configuration (several threads using several configurations) processing.

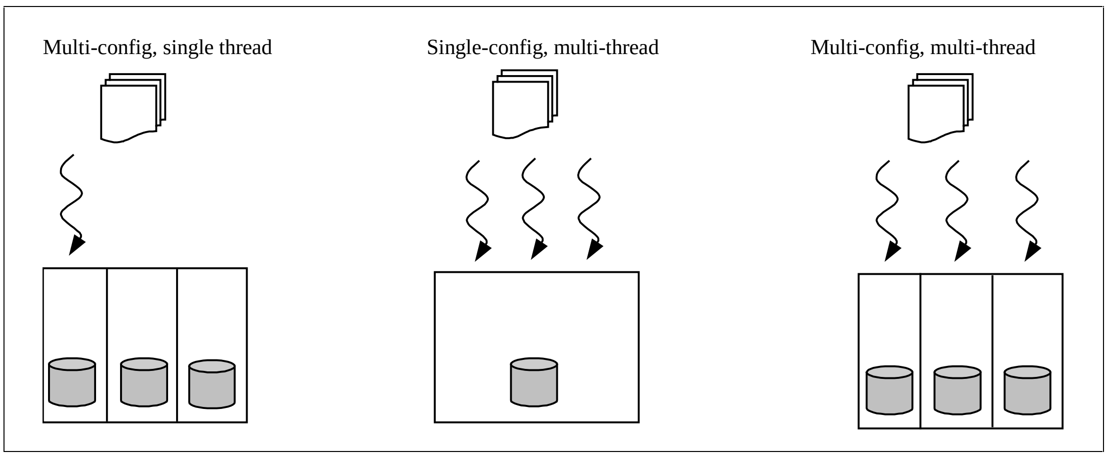

Figure 3: Multi-thread Capabilities

NameTag Configuration

Although the NetOwl Extractor engine executes the text processing phases, the content of the configuration determines the exact characteristics of processing. The lexicon and pattern rule base define what the engine recognizes, a template (tag) specification and action definitions define what the engine extracts, and the processing classes define the distinct processing phases that the engine performs. Global parameters provide run-time control of processing.

The core processing of the **NameTag Configuration** is proper name recognition. Proper name recognition requires specialized linguistic knowledge about the structure or composition of each type of name. For example, person names usually have first names and last names, with optional middle names or initials. Proper name recognition also requires knowledge about how names appear in free text. This knowledge consists of contextual clues about how each type of name may appear. For example, person names may have professional titles or descriptions preceding or following the name.

Proper name recognition also requires a substantial list of proper names in the lexicon in order to identify household names that do not provide the structural indication of what kind of name it is. For example, most country names, such as “France”, have no internal composition indicating the name is a country. Also, many types of text may not provide proper descriptions or contextual clues about common or assumed proper names. For example, business news articles may assume that the reader knows what the acronym “NYSE” stands for.

Proper name recognition also requires the identification of aliases, or shortened variations of the full proper name. For example, many organizational names have acronyms, person names have initials, and corporate names tend to exclude the official corporate designator, such as “Corp.” The **NameTag Configuration** contains alias generation rules that are applied to recognized names.

A major problem with name recognition is the ambiguity of names. Person, place, and organizational names can be composed of the same words. For example, the word “Jordan” can be a first name, a last name, a country name, or part of an organizational name. Different types of names can appear in text in similar ways. For example, person, place, and organization names can be the subject of communicative verbs, such as “said”.

The **NameTag Configuration** addresses this major problem by executing a rule competition phase to select the most probable interpretation for a name. The rule competition uses the numeric weight of each rule, factors in the length of each interpretation, and sums these values according to the type of tags. High rule weights indicate strong evidence, low rule weights indicate weak evidence, and negative rule weights indicate counter-evidence. The competition also considers interpretations based on proper name entries in the lexicon, thus allowing dynamic pattern matching results to compete against static name look-up. The rule competition eliminates interpretations with negative sums, and selects the interpretations with the highest sums.

Visual Extractor

NetOwl Extractor provides a development graphical user interface (GUI) called Visual Extractor, which is implemented in Java. The GUI is totally independent of the configuration, just as the NetOwl Extractor engine is. Visual Extractor provides a project workspace that includes a source file view, a settings view, and a configuration object view. The workspace can include multiple configurations and parameter settings to allow greater flexibility while one evaluates the performance of the system. Visual Extractor can export its settings as a batch parameter file or as programs in C or Java. Visual Extractor has an extensible extraction display, allowing many views of the extraction results. **Figure 4** shows a screen shot of Visual Extractor.

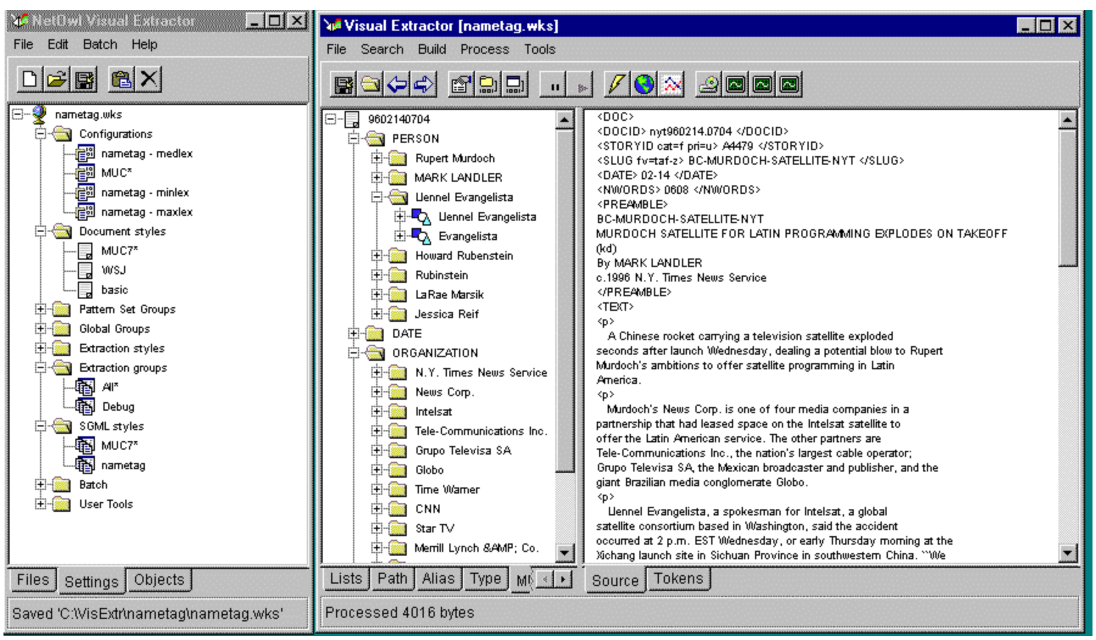

Figure 4: Visual Extractor

NAMED ENTITY CUSTOM CONFIGURATION

Without modification, NetOwl Extractor’s **NameTag Configuration** identifies all of the types of names and phrases included in the Named Entity task specification. In fact, the **NameTag Configuration** identifies substantially more expressions and has a richer tagging scheme. However, the Named Entity tag specification does contain a few significant differences to the product specification of NameTag, such as tagging numeric and temporal ranges as one tag where as the **NameTag Configuration** breaks them into two separate tags. Also, the Named Entity task required systems to specially tag the header information in the New York Times documents, such as the `SLUG` word. For these reasons, IsoQuest created a Named Entity custom configuration based on the core **NameTag Configuration**.

The NE configuration contained a small number of custom patterns to handle the document style and tagging differences. The configuration also included some domain-specific lexical entries for the training set. Using the standard API, users could write a small program to map the extracted tags into the NE tags and generate the SGML format themselves. Instead, IsoQuest added a small action script to the NE Configuration to perform this mapping, and thus allow the NE tags within Visual Extractor and the direct output of SGML from the standard API. **Figure 5** shows the comparison between the **NameTag Configuration** and the NE configuration in terms of total size, lexicon size, and pattern rule base size.

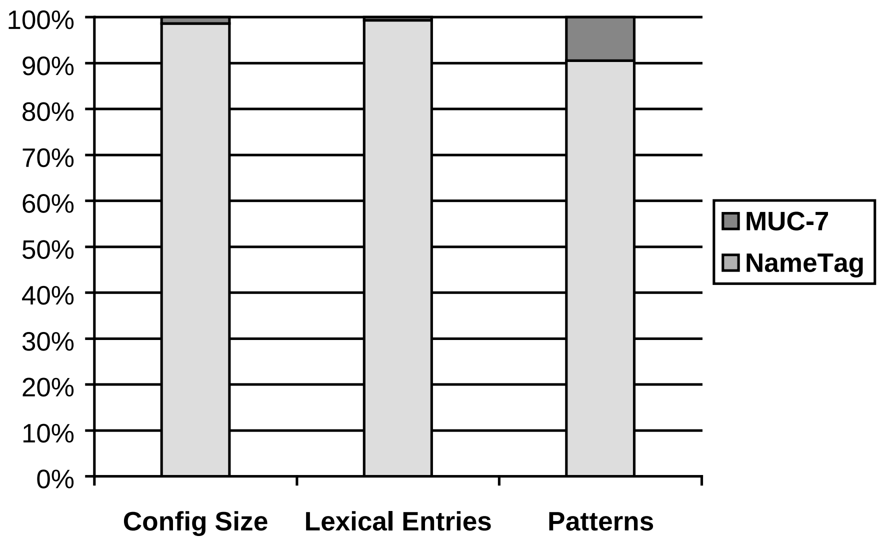

Figure 5: NE vs. NameTag Configuration

Tag Mapping

Overall, the mapping of **NameTag Configuration** tags to the Named Entity tags is straight forward. However, due to the complexity of the Named Entity specification (20 pages plus the FAQ), the mapping had to account for a substantial number of exceptions and special cases. Inconsistencies, ambiguities, and unspecified cases in the MUC-7 specification impact the effectiveness of this mapping, and ultimately the final results. **Table 1** summarizes the tag mapping.

NameTag	NE	Comments
NUMERIC	NUMEX	only PERCENT and MONEY
PLACE	ENAMEX (LOCATION)	except some roadways, …
PERSON	ENAMEX (PERSON)	Except “X report”
ENTITY	ENAMEX (LOCATION)	Just airports, camps, forts, …
ENTITY (FACILITY)	none	Except museums, lodges, …
ENTITY (PUBLICATION)	ENAMEX (ORG)	Except when referring to the paper
ENTITY	ENAMEX (ORG)	Everything else, except …
TIME (TIMEOFDAY) (DATE)	TIMEX (TIME) (DATE)	Direct mapping.
TIME (TEMPORAL)	TIMEX (DATE)(TIME)	DATE and TIME were assigned based on the time unit. Some temporal expressions were filtered.
OTHER	none	Named events, products, equipment and other non-entities
MISC	none	E-mail addresses, others

Table 1: NE Tag Mapping

Named Entity Custom Patterns

As described above, the Named Entity configuration included a few special patterns to account for the special document style as well as minor differences in the tagging scheme. This following list explains the basic types of custom patterns and provides examples of them.

- *Document style patterns* –Seven patterns executed before the name recognition patterns and created special tags around portions of the document. For example, one pattern identified the SLUG word inside of the PREAMBLE section so that both occurrences of the SLUG word could be processed identically. Another pattern created a DATE tag in the NWORD section when it contained a month/day expression. Other patterns operated on the PREAMBLE and TRAILER sections.
- *Publication patterns* – Five patterns executed after the name recognition patterns and modified the ENTITY tags for publications if they appeared in certain contexts indicating the physical paper rather than the organization. For example, the phrase “edition of X” indicates that “X” is the physical paper, so it should not be tagged.
- *Compound tags* – Ten patterns executed after the name recognition patterns and created a compound tag for a sequence of individual tags. For example, these patterns would create a single tag for numeric and time expressions of the form “from X to Y”. Another pattern created a single PERSON tag for expressions in the form of “first name and PERSON”.
- *Executive departments* – Three patterns executed after the name recognition and created ENTITY tags for governmental departments appearing as title modifiers. For example, these patterns create a ENTITY tag for “transportation” in the phrase “transportation secretary”. The **NameTag Configuration** does not treat these departmental references as true proper name mentions.
- *Other* – Five patterns executed during the name recognition patterns and created additional tags for other miscellaneous cases. For example, one pattern identified certain names of the form “X OrgNoun of Place” and tried to split them into an ENTITY and PLACE tag according to the NE task specifications.

Named Entity Application Program

NetOwl Extractor provides a simple, flexible API that allows users to open configurations, specify run-time parameters, load and process documents, and extract the results. The API also includes a built-in SGML writer that can output the extracted tags in SGML format. The following listing is the complete application program for the NE task. Simple programs such as this provided with the product make it simple for application programmers to quickly begin their own development.

```
int main(int argc, char ** argv) {
  NEcfg cfg;
  NEdoc doc;
  NEextr extr;
  NEsgml sgml;
  int bytes = 0, tokens = 0, dcnt = 0, result, error = 0, docSec;
  char buf[256];

  try {
    if ((result = NEinit()) != E_SUCCESS) throw result;
    if ((result = NEopenConfiguration(argv[1],&cfg)) != E_SUCCESS) throw result;
    if ((result = NEcreateDocument(cfg,&doc)) != E_SUCCESS) throw result;
    if ((result = NEdocLoadParameters(doc,argv[2],1,&docSec,&extr,&sgml)) != E_SUCCESS) throw result;
    unlink(argv[3]);

    if ((result = NEdocOpenFile(doc,argv[4])) != E_SUCCESS) throw result;
    while ((result = NEdocLoadRegion(doc,docSec,&tokens)) == E_SUCCESS && tokens > 0) {
      dcnt++;
      bytes += NEdocBytesLoaded(doc);
      if ((result = NEdocProcess(doc)) != E_SUCCESS) throw result;
      if ((result = NEsgmlWriteFile(sgml,argv[3],1)) != E_SUCCESS) throw result;
      NEdocClear(doc);
    }
    NEdocClose(doc);
    NEcloseExtractor(extr);
    NEcloseSGML(sgml);
    NEcloseDocument(doc);
    NEcloseConfiguration(cfg);
  }
  catch (int error) {
    NEgetErrorString(error,buf,256);
    fprintf(stdout,"Error %d: %s\n",error,buf);
  }
  NEshutdown();
  return error;
}
```

TEST RESULTS AND ANALYSIS

IsoQuest submitted two runs for the formal NE test. The *Official* run used the full pattern rule base to perform the maximum analysis, achieving the best results at the slowest speed. The *Optional* run used about 20% of the rules to perform the minimum analysis, achieving lower performance at the greatest speed. The performance measures and timing information are shown in **Table 2**. The time includes initialization and SGML output, and was computed on a Pentium II 300 Mhz processor. As another reference point, a Pentium 133 Mhz laptop performed these two runs at 140 Meg/hour and 190 Meg/hour.

Test Run	Recall	Precision	F-Measure	CPU Time (seconds)	Speed (Meg/hour)
Official	90	93	91.60	3.6	382
Optional	74	93	82.61	2.7	513
ALLCAPS	78	96	81.96	4.9	279

Table 2: NE Test Results

IsoQuest performed an additional experimental run using the case-insensitive configuration against the upper-case version of the test data. For the *ALLCAPS* run, the configuration was optimized for high precision. Since the MUC-7 task specification includes case-sensitive tagging rules, the upper-case version of the test data should be manually re-tagged. This would most likely improve the case-insensitive results.

The size of the NetOwl Extractor process depends on the size of the configuration, which largely depends on the size of the lexicon. As an experiment, IsoQuest ran the formal test documents against three configurations containing different number of lexical entries. The results of this experiment are summarized in **Table 3**. The size of the lexicon has a small effect on the extraction performance since the pattern rule base relies on the content of the lexicon to identify proper names and phrases.

Lexicon Size	Approximate Entries	Process Size (Meg)	F-Measure
Maximum	110000	7.0	91.60
Medium	25000	4.0	91.45
Minimum	9000	3.7	89.13

Table 3: Lexicon Size Experimental Results

Error Analysis

The *Official* run performed significantly worse than the dry run test in September, approximately 3 points lower in F-measure. This drop in performance is almost entirely due to the fact that the domain of the formal test documents (satellites) was different from the training and dry run documents (airlines). Strangely, the Named Entity task is defined as a domain-independent task, yet the MUC committee selected samples of New York Times articles that focused on particular domains. This selection process greatly influenced the results of the task and substantially diminished the value of the formal test. The formal test failed to achieve its objective of measuring domain-independent name entity identification. Instead, the formal test measured how well systems perform on a new domain and how well the airline domain prepared systems for the undisclosed satellite domain. If the samples were randomly selected, the formal test would have measured how well systems perform in general and how well systems improved from the dry run.

The performance measures for the training set and dry run set are shown in comparison to the formal test results in **Table 4**. The formal test results (*) have been adjusted to account for a major inconsistency in the manual tagging of a certain type of temporal expression, which is irrelevant for this analysis.

Test Set	Recall	Precision	F-Measure
Training (9/97)	97	97	97.27
Training (3/98)	98	99	98.27
Dry run (9/97)	94	97	95.35
Dry run (3/98)	94	98	96.28
*Formal (3/98)	92	93	92.56

Table 4: Training and Test Set Comparison

A drop in performance is typical behavior for the **NameTag Configuration** when it encounters a document style that has fewer descriptive contextual clues than other rich sources such as the Wall Street Journal. The **NameTag Configuration** does not rely on static lists of proper names although it can accommodate them. The New York Times document style used in MUC-7 had significantly fewer clues especially for organization names. **Table 5** shows the effect of the airline-specific lexicon modifications on the dry run set. The recall of ORGANIZATION tags increased 9 points.

Test set	ORGANIZATION (REC/PRE)	PERSON (REC/PRE)	LOCATION (REC/PRE)
Dry run (unmodified)	85/99	94/98	96/98
Dry run (3/98)	94/99	93/98	96/97
Formal (3/98)	87/89	94/97	93/95
Formal (modified)	93/95	96/99	96/95

Table 5: ENAMEX Performance Improvement Due To Lexicon Modifications

Configurations of NetOwl Extractor are customizable and users are encouraged to make minor adjustments to their configuration to optimize their performance. A quick analysis of the NE formal test documents uncovered various common and problematic names from the satellite industry. As an experiment, IsoQuest added a small number (42) of custom entries to the Named Entity configuration and re-ran the formal test documents, as summarized in **Table 6**. Lexical entries can either be positive or negative indicators of a tag type. **Table 5** shows the effect of the lexicon modifications on the formal test set. The recall of ORGANIZATION tags increased 6 points. The precision also increased 6 points due to the elimination of ORGANIZATION tags for some common non-organizations.

NAME	TAG	POLARITY	NAME	TAG	POLARITY
AMSC	ENTITY	+	ARIANE	ENTITY	+
ASIASAT	ENTITY	+	BBN	ENTITY	-
CANALSATELLITE	ENTITY	-	CHINASAT	ENTITY	+
CLIPPERS	ENTITY	+	COLUMBIA	OTHER	+
COSAT	ENTITY	+	DBS	ENTITY	-
DELTA	OTHER	+	DIGITAL TV	OTHER	+
EARTH	PLACE	+	EDDIE BAUER	ENTITY	+
ENDEAVOUR	OTHER	+	EROS	PLACE	+
HUGHES	ENTITY	+	MARS	PLACE	+
MA QIMIN	PERSON	+	MCCLEESE	PERSON	+
MIR	OTHER	+	MISSION CONTROL	ENTITY	+
MOON	PLACE	+	MURDOCH	PERSON	+
NI	ENTITY	-	PANAMSAT	ENTITY	+
PAS	ENTITY	+	PRIMESTAR	ENTITY	+
RADIOSAT	ENTITY	+	RCA	ENTITY	+
REPUBLICANS	ENTITY	-	RINEY	ENTITY	+
ROCKETDYNE	ENTITY	+	ROCKETS	ENTITY	+
SATURN	OTHER	+	SPACE	ENTITY	-
SPACE FLYER UNIT	OTHER	+	SPACE SHUTTLE	ENTITY	-
SPACE STATION	ENTITY	-	STU	ENTITY	-
XICHANG	PLACE	+	X-33	OTHER	+

Table 6: Custom Lexical Entries

SYSTEM WALKTHROUGH

MUC-7 selected document 9602140704 from the Named Entity formal test as the system walkthrough example. IsoQuest's official run achieved an F-measure of 89.15 (85 recall, 94 precision) on this document. **Table 7** summarizes the errors made by NetOwl Extractor in this document. This document typifies the overall performance since most of the errors are due to household or domain-specific proper names. Seven of the errors involved the household name "Murdoch". NetOwl Extractor correctly determines that the phrase "Murdoch's News Corp." is a possessive construct between a person alias and a company name, but it erroneously determined that "MURDOCH SATELLITE" was an ORGANIZATION. Due to this mistake, NetOwl Extractor does not tag three other mentions of "Murdoch" as a person alias.

Tag	Error	TYPE	Key	Response
ENAMEX	missing	PERSON	MURDOCH	
ENAMEX	missing	PERSON	MURDOCH	
ENAMEX	missing	PERSON	MURDOCH	
ENAMEX	missing	LOCATION	Xichang	
ENAMEX	missing	ORGANIZATION	Viacom	
ENAMEX	missing	ORGANIZATION	Home Box Office	
ENAMEX	missing	ORGANIZATION	Turner Broadcasting System	
ENAMEX	missing	PERSON	Murdoch	
ENAMEX	missing	PERSON	Murdoch	
ENAMEX	spurious	ORGANIZATION		MURDOCH-SATELLITE
ENAMEX	spurious	ORGANIZATION		MURDOCH SATELLITE
ENAMEX	spurious	ORGANIZATION		CNN

Table 7: System Walkthrough Errors

NetOwl Extractor missed two other household names: "Viacom" and "Home Box Office". NetOwl Extractor tagged "Turner Broadcasting System" as an OTHER tag and thus filtered it from the NE output. NetOwl Extractor failed to tag "Xichang" as a LOCATION, due to the lack of domain knowledge about launch sites. The phrase "Xichang launch site" would match an existing NameTag pattern if "launch site" was added as a location noun. Finally, NetOwl Extractor identified “CNN” as an ORGANIZATION but failed to determine that the phrase “beam MTV, CNN and other channels” indicates that “CNN” is a channel and not an organization, according to the Named Entity task specification.

CONCLUSION

IsoQuest successfully used its commercial software product NetOwl Extractor for the Named Entity task in MUC-7. With minor modifications to the **NameTag Configuration**, IsoQuest performed the NE task with high performance, although lower than the intermediate results. IsoQuest demonstrated that the drop in performance was mainly due to the document style combined with the change in domain of the formal test documents, and showed how to improve performance with simple additions to the lexicon. IsoQuest demonstrated the ease of programming with NetOwl Extractor’s API and demonstrated its high speed and low memory.

ACKNOWLEDGEMENTS

Greg Roberts provided valuable support for IsoQuest’s MUC-7 effort, including analyzing system performance and interpreting the Named Entity task specification.

IsoQuest, a wholly-owned subsidiary of SRA International Inc., is a leading provider of information discovery products that automatically extract, analyze, index, categorize and summarize text documents. IsoQuest develops, markets and supports the NetOwl Family of Intelligence Discovery Products, including NetOwl Extractor, NetOwl for Electronic Publishing and NetOwl for Classifieds. NetOwl helps corporations manage electronic information and allows end users to know more by reading less. Headquartered in Fairfax, Virginia, IsoQuest serves a global customer base including IBM, Thomson Corporation, OCLC, EDGAR Online, NewsEdge Corporation and Knight Ridder New Media. More information about IsoQuest is available at www.isoquest.com.